ЎЗБЕКИСТОН РЕСПУБЛИКАСИ АЛОҚА, АХБОРОТЛАШТИРИШ ВА ТЕЛЕКОММУНИКАЦИЯ ТЕХНОЛОГИЯЛАРИ ДАВЛАТ ҚЎМИТАСИ

ТОШКЕНТ АХБОРОТ ТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ

Қўлёзма ҳуқуқида

УДК 004.357

РАХИМОВ МЕХРИДДИН ФАЗЛИДДИНОВИЧ

Мультимедиа тизимларида ахборотни жойлаштириш, сақлаш ва саралаш амалларини оптималлаштириш

5А330203 – “Амалий информатика”

Магистр

академик даражасини олиш учун ёзилган дессертация

Илмий рахбар:

т.ф.д, проф. Мусаев М.М.

Тошкент – 2014

М У Н Д А Р И Ж А

Кириш

Ахборот–телекамуникация технологиялари ва илм-фаннинг ривожланишига бўлган эътибор давлатимизнинг мустақилликка эришган дақиқалардан бошлаб кучая бошлади. Ўзбекистон Республикасининг «Ахборотлаштириш тўғрисида»ги қонуни асосан [1], давлат сиёсатининг ахборотлаштириш соҳасидаги вазифаси ахборот бозорида маҳсулотларни, хизматларни ва ахборот технологияларни тартибга солиш, дастурий махсулот ишлаб чиқаришни рағбатлантириш, мутахасислик бўйича кадрлар таёрлаш ва уларнинг сифатини ошириш ва албатта илмий изланишга бўлган талабларни рағбатлантириш кабилардан иборат эди.

Ўзбекистон Республикаси Президентининг “Замонавий ахборот-коммуникация технологияларини янада кенгроқ жорий қилиш ва ривожлантириш чора-тадбирлари тўғрисида”ги қарорида [2] замонавий ахборот технологияларидан фойдаланиш, кампьтер техникаси ва телекамуникатся воситаларини иқтисовий ҳамда хаётий оммага тадбиғ этиш норматив хужатлари белгилаб берилди. Хозирда ахборотлаштириш сохасидаги давлатимиз сиёсати миллий ахборот тизимини яратишда кампьютер технологияси ривожланиш тенденсясининг хозорги замонавий ҳолатини хисобга олган ҳолда ахборот технологиясини ва тизимини ташкил этиши лозим. Президент Ислом Каримовнинг Ўзбекистон Республикаси Конституцияси қабул қилинганининг 18 йиллигига бағишланган « Мамлакатимизни модернизация қилиш йўлини изчил давом эттириш – тараққиётимизнинг муҳим омилидир» номли маърузаларида, ахборот технологиялари махсулоти ва хизмат кўрсатиш доирасининг сифат даражаси келажакда мамлакатимизни модеринизациялашда жуда муҳим аҳамиятга эга эканлигини таъкидлаб ўтгандилар [3].

Миллий ахборот тизимининг ривожлантиришнинг яна бир муҳим йўналишларидан бири мультимедия тизимларида маълумотларни хотирага ёзиш, сақлаш ва ўқиш амалларини оптималлаштириш ва тасвирларни

қайта ишлаганда процессор унумдорлигини оширишдаги муаммоларни бартараф этиш йўллари ҳисобланади.

Мавзунинг долзарблиги. Ахборот технологияларинг охирги ўн йиллик ривожланиш даврида мультимедиа тизимларида ҳам сезилар даражада таъсир кўрсатти. Айниқса, мультимедиа ахборотининг хотира билан боғлиқ амалларидаги хатоликларни бартараф этишнинг дастурий ҳамда техник воситаларга бўлган эҳтиёж янада кучайди. Мультимедиа тизимидаги ахборот турлари: матн, тасвир, нутқ, анимация ва видео кабиларниг хотирадан катта жой эгаллаши унинг қайта ишлаш амалларини процессорга юклаганда ва олинган натижаларни компьютер хотирасига ёзиш жараёнларининг кўп вақт талаб қилиши ва хотира русурсларининг чекланганлиги янги усулларни яратишни ва ананавий қайта ишлаш алгоритмларини такомиллаштиришни талаб қилмоқда. Процессор унумдорлигини оширишда ҳозирги вақтга келиб анаъна бўлиб қолган машхур фирмаларнинг процессорларнинг транзисторлар сонини ошириш борасидаги мусобоқаси ниҳояланиб қолди. Эндиликда процессор ишлаб чиқарувчилар транзисторлар сонини эмас, балким процессорнинг ядролар сонини оширишга киришиб кеттилар. Албатта бундан келиб чиқадики яратиладиган дастурий таъминот ва воситалар кўп ядроли процессорларга мўлжаллаб ишлаб чиқарилиши лозим. Акс ҳолда техник таъминотнинг ривожланиб бориши унумдорлик даражасининг ошиши учун етарлича ҳисобланмайди [16].

Айниқса тасвир маълумотларини рақамли қайта ишлаганимизда катта хажмдаги маълумотлар оқимига дуч келамиз ва хотирага мурожаат қилиш буйруқлари кўпайиб, қайта ишлаш вақтининг чизиқсиз ошиб боришига сабаб бўлади. Шунинг учун тасвирларни рақамли қайта ишлаганда биринчидан, параллелаштириш алгоритмларини амалиётга тадбиқ этиш ва процессорда маълумотларни қайта ишлаганда оқимларга ажратиш усулларидан кенг фойдаланиш, иккинчидан, хотирага мурожаат

қилиш қадамлар сонини камайтириш процессор унумдорлигини оширишда яхши натижа беришини кафолатлайди [4, 5, 6].

Аммо паралеллаштириш алгоритмларини яратиш муаммони ечими ҳисобланмайди, шунинг учун ҳар қандай масалани параллел ечишда унверсал ечим бўла олмайди. Бундан ташқари тизимли дастурлашда параллелаштиршни амалга ошириб берувчи бир қанча дастурий воситалар ҳамда усуллар яратилган. Бундай махсулотларга созлагичлар, маҳсус шаклга келтиргичлар ва маҳсус кутубхоналар яратилган [12, 13, 19-30].

Ишнинг мақсади ва вазифалари. Мультимедиа тизимларида жойлаштирилган ахборотни тезкор ўқиш жараёнларни оптимал йўлларини излаб топиш, хотира ресурслари билан боғлиқ муаммоларни бартараф этиш ва тасвирларни рақамли қайта ишлашда кўп ядроли процессорлар унумдорлигини ошириш учун мўлжалланган параллелаштириш алгоритмларининг оптимал усулларини яратиш. Юқорида айтиб ўтилган мақсадга эришиш учун, бу иш доирасида қуйидаги **вазифалар** қўйилди ва бажаририлди:

- мультимедиа тизимларида маълумотларни ёзиш, сақлаш ва ўқиш амаллари адабиётлар манбаидан кўриб чиқилди ва ундаги тасвирларни рақамли қайта ишлаш ва хотирага жойлаштириш усулларидаги муаммоларни таҳлил қилиш;
- тасвирларни сиқишда вейвлет-жараён орқали кетма-кет қайта ишлаш алгоритмлари билан танишиш ва амалиётда фойдаланиш;
- кетма-кет алгоритмлардан фойдаланган ҳолда тасвирларни вейвлет-жараён орқали қайта ишлаганда процессор ресурсларини қайси амаллар кетма-кетлиги кўпроқ банд қилишлигини аниқлаш;
- алгоритмларнинг параллелаштирилган усулларидан фойдаланган ҳолда маълумотни ўқиш ва ёзиш қисмларни оптимал хажмини аниқлаш;

- алгоритмларнинг параллелашган қисмини яратишда OpenMP компиляторининг параллелаштириш директиваларидан тўлиқ фойдаланиш;
- параллелаштирилган алгоритмларни кўп ядроли процессорларда тасвирлар мисолида тажрибалар ўказиш ва олинган натижаларни таҳлил қилиш;
- тажрибалар натижасида келиб чиққан ҳолда тасвирларни рақамли қайта ишлашда мақбул параллелаштириш усулларини танлаб олиш.

Тадқиқот объекти ва предмети. Тадқиқот объекти - тасвирларни параллел қайта ишлаш ва кўп ядроли процессорлар. Тадқиқот предмети – мультимедиа тизимларида хотира билан боғлиқ амаллар, параллел алгоритмлар яратиш йўллари, параметрлари, уларнинг оптималлаштиришга тасири, дастурий параллелаштириш кутубхоналари, кўп оқимли иловаларни ишлаб чиқиш ва унумдорлигини аниқлаш учун мўлжалланган махсус дастурлар.

Тадқиқот услубияти ва услублари. Илмий ишда дастурлаш параллелаштириш усуллари, алгоритмлар назарияси, математик ва статистик таҳлил, тасвирларни қайта ишлашда вейвлет жараёнлардан фойдаланиш.

Тадқиқот натижаларининг илмий жихатдан янгилик даражаси. Тасвирларни сиқишда вейвлет-жараёнини қўллаган ҳолда, амаллар кетма-кетлигини оптимал хажмдаги оқимларга ажратиш, параллелаштириш алгоритмларидан унумли фойдаланиш тажриба асосида оқимларни оптимал хажмини аниқлаш процессор ядролар сонининг унумдорликка кўрсатадиган таъсирини аниқлаш. Ҳозирги пайта тасвирларни қайта ишлашда кўпгина дастурларида бу муаммоларнинг ечими охиригача ечилмаган.

Тадқиқот натижаларининг амалий аҳамияти ва тадбиғи. Тасвирларни вейвлет-жараён орқали қайта ишлаш сиқиш амалида яхши

натижа беради, яъни хотира ресурсларининг етишмовчилиги муаммоларини ечими бўлади. Бундан ташқари қуйидаги параллелаштириш алгоритмларидан фойдаланиш тасвирларни рақамли қайта ишлаш тезлигини оширди ва унумдорлик даражаси процессорнинг оқимлар сонига яқинлашди. Тажрибалар натижасида оқимларни қандай параметрлари унумдорлик даражасини оширилиши мумкинлиги аниқланади ва тасвирларни рақамли қайта ишлашда оқимларни мос хажмлари қўлланилади.

Иш тузилиши ва таркиби. Қуйидаги илмий иш кириш, учта боб, хулоса, 30 та фойдаланилган адабиётлар рўйхати ва битта иловадан ташкил топган. Келтирилган иш 78 бетдан, 4 та жадвал ва 27 та расмдан иборот.

I боб. Мультимедиа тизимлари ва хотирага маълумотларни ёзиш, сақлаш ва ўқиш амалларидаги муаммолар таҳлили

1. Мультимедиа тизимларининг таркиби ва вазифалари

Мултимедия технологияси 20 асрнинг 90-йилларидан тез суратлар билан ривожлана бошлади. 21 асрга келиб ушбу технологиялар мултимедия амалий дастурлари яратиш, мултимедия махсулотлари ва хизматлари асоси бўлиб қолди. Мултимедия компютер техникасида муҳим ўрин эгаллади. Компютер техникасининг ривожига мултимедия технологияларининг таъсири катта бўлди, малумотларни хотирада сақлашнинг янги технологиялари яратилди, микропросессорлар тезлиги анчагина ошди.

- **Мултимедиа [М-медиа]** атамаси "*мульти*" — кўп ва "*медиа*" — муҳит, ташувчи, хабар воситаси маъноларини билдирувчи сўзлар бирикмасидан ташкил топган бўлиб "кўп муҳитлик" деб таржима қилиш мумкин.
- ***Мултимедия*** — бу компютер тизими орқали матн, товуш, видео тасвир, график тасвир ва анимация каби элементларни бирлаштирувчи замонавий компютер ахборат технологиясидир.
- ***Мултимедия*** — бу матн, график, анимация, видео, товуш ва нутқ каби берилганларни компютерга киритиш, қайта ишлаш, сақлаш, узатиш ва тасвирлаш каби амалларни бажарувчи технологиялар мажмуйидир.
- ***Мултимедиа-қурилма*** — бу товуш, графика ва вдео кўринишли маълумотлар билан ишлашга мўлжалланган компютер қурилмаларидир.
- ***Мултимедиали компютер*** — бу мултимедиа технологиялари асосида ишлай оладиган техник ва дастурий таъминотга эга бўлган компютердир.

Мултимедиа тизими – бу аудио, видео, анимация ва график кўринишидаги маълумотларни ҳосил қилиш, қайта ишлаш, сақлаш ва чиқариш каби амалларни бажаришга қаратилган мултимедия воситалари ва технологиялари мажмуидир. Мультимедия тизимининг асосий характеристикаси бу фойдаланувчига аудио, видео, анимация, графика кўринишдаги маълумотларни юқори сифатда намаён этишдир. Шу билан бирга уларнинг озаро боғлиқликда ва бир бирини тўлдирган ҳолда фойдаанинш имкониятининг мавжудлигидир. Масалан, видео филмларда аудио ва матнли маълумотларни ишлатиш; мусиқа ва қўшиқ матнини бирга ишлатиш орқали ҳосил қилинадиган "караока"; графикли малумотларни мусиқали намойиши бўлмиш "слайд шоу"лар.

Бугинги кунда мултимедия маълумотларни тасвирлашнинг асосий усуллари қуйидагилардан иборат [17]:

- Аудио;
- Видео;
- Матн;
- Анимация;
- Тасвир;
- Интерактивлик (мулоқатли).

Мултимедия тизими техник ва дастурий таъминотлардан ташкил топади. Техник таъминот таътаркибига:

- **График акселаторлар (тезлатгичлар)**. Замонавий видео карталарнинг барчаси график акселатор ҳисобланади.
- **CD-ROM/RW,DVD-ROM/RW** -оптик қурилмалар.
- **Товуш карталари ва товуш тизимлари**. Товуш тизимлари — бу савбуферлар (кучайтиргичлар) бўлиб, улар ўзларига бир неча товуш калонкаларини бирлаштирган ҳолда товуш эшитилишларни бошқаради.
- **Тюнерлар**– бу телевизион ва видео магнитафон сигналларини қабул қилиш ва кўрсатиш имконига эга бўлган **ТВ-тюнер** лар ва радио сигналларни қабул қилувчи **FM-тюнерлардир.**

- **Колонкалар** — товуш эшитиш қурилмалари. Улар **пассив** ва **актив** турларга бўлинади. **Пассив калонкалар** товуш карталарининг ички овоз кучайтиргичлари ҳисобига ишлайди. **Актив калонкалар** ўзларининг товуш кучайтиргичларига эга бўлади, шунинг учун уларда товуш эшитилиши уларда кучлироқ ва сифатлироқ бўлади.
- **Микрофон.** Бу қурилма овозли маълумотларни компютер ҳотирасига киритиш қурилмаси. Микрофон овоз картаси қурилмасига уланади.
- **МИДИ-клавиатура.** МИДИ стандарти асасида мусиқа ҳосил қилишга мўлжалланган қурилма бўлиб, товуш картанинг МИДИ разёмига уланади. Бу синтезатор клавиатурасига ўхшаган клавиатура бўлиб унда мусиқали сигналлар компютер сигналларига ўтказиб беради.
- Бошқа компютерни ташқи дунё билан боғлайдиган қурилмалар, масалан, рақамли фотоаппаратлар ва видеокамералар, МП3-плеейерлар каби қурилмалар киради.

Дастурий таминотга эса мултимедия маълумотларини қайта ишловчи ва техник таъминотни бошқарувчи дастурлар киради. Масалан Windows Movie Maker, Soubde Forge, MX-Flash ва ҳаказо.

Ҳозирги даврда мултимедия технологияси Web-саҳифалар ва Web-дастурлар яратишда кенг кўламда фойдаланилмоқда.

Мултимедия қуйидаги турларга бўлинади:

- **Гипермедиа [G-Медиа]** – бу графика, овоз, видео, матн кўринишидаги маълумотларни гиперматн кўринишида тасвирлаш орқали озаро боғланган мултимедия маълумотларнинг чизиқли бўлмаган муҳитидир;
- **Интерактив мултимедия** – бу видео тасвирларни ва овозли маълумотларни мулоқат режимида бошқариш имконини таминлай оладиган мултимедия тизимидир;
- **Жонли видео** – бу мултимедия тизимининг реал вақт мабойнида ишлаш нуқтаий назаридан бериладиган ҳарактеристикасидир.

Мултимедия тизимлари – ишлатилиш мақсадига кўра уйда, тижоратда ва умумий мақсадларда ишлатиладиган тизимларга ажратилади.

Уй мултимедиятизимлари вақтни завқли отьказиш ва дам олиш мақсадида ишлатишга мўлжалланган. Уй мултимедия тизими – бу мултимедия воситаларининг ўз аро бирлашмаси бўлиб, унинг таркибига уй кинотеатри, мултирум, телевидение, медиа сервер ва аудио тизим кабилар киради. Масалан уй мултимедия тизимига мултирумни мисол қилиш мумкин.

Мультирум (МултиРоом) – бу ауди ва видео сигналларларни кўп зонали тақсимот қилиб берувчи қурилмадир. Унинг ёрдамида уйнинг овоз чиқарувчи ва видео кўрсатувчи қурилмалар ўрнатилган ҳоналаридан ташқарида ҳам, масалан, овқатланиш ҳонасида, ванна ҳонада, кутубхонада, гаражда ва ҳовлида мусиқа эшитиш ва филмлар кўриш мумкин. Мультирум тизимини бошқариш деворга ёки столга ўрнатилган стационар тугмачали панеллар, масофали бошқариш пултлари ёки сенсорли пенеллар тизими орқали амалга оширилади.

Медия сервер — бу мултимедиа кўринишидаги маълумотларни ўзида сақловчи қурилмадир. Бу қурилмада видео филмлар, аудио ва фото материаллар сақланади. Медия сервер фақат видео материаллар ёки слайд-шоулар кўрсатиб қолмасдан, яна уларни бир пайтда бир неча видео ва аудио манбаларга транслясия қилиш имконини ҳам яратади.

Тижорат ёки умумий мақсадларда ишлатиладиган мултимедия тизимлари иш жараёнини оптималлаштириш, ҳодимларни ўқитиш, видео намойишлар, ва видео конференсиялар ўтказишга мўлжалланган. Улар маълумотларни қайта ишлаш марказларида, диспетчерлик хизматларида, ўқув заллари ва хоналарида кенг кўламда ишлатилмоқда.

Ҳозирда мултимедиянинг қуйидаги технологияларидан кенг фойдаланиб келинмоқда:

- Видеоконференсия;
- Мултимедияли ўқитиш дастурлари;

- Електрон газеталар ва китоблар;
- Овозли ва видео почта;
- График дизайн воситалари.

Мултимедиянинг бу келтирилган технологияларидан инсон фаолиятининг жуда кўп соҳаларида кенг фойдаланилмоқда. Бунга мисол тариқасида қуйидаги соҳаларни келтириш мумкин:

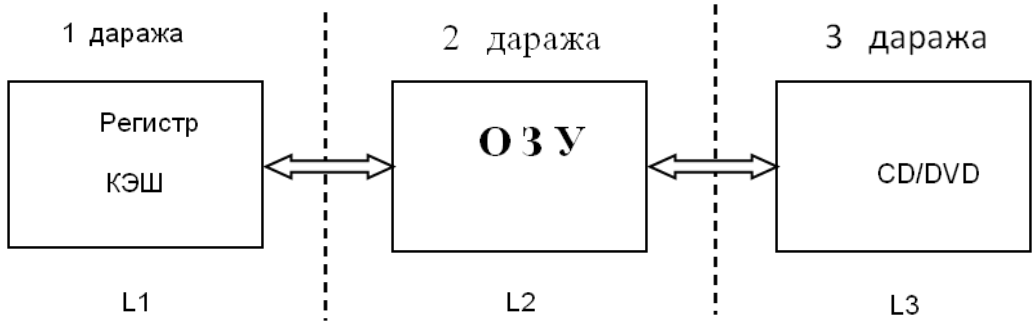
1. Компьютер ўйинлари яратиш соҳасида;
2. Таълим соҳасида;
3. Медицина соҳасида;
4. Тижорат ва менеджмент соҳаларида;
5. Ҳарбий соҳада;
6. Ахборот таъминоти соҳасида (электрон маълумот олиш; электрон энциклопедия; бадиий луғатлатлар ва таржимон дастурлар);
7. Санъат соҳасида;
8. Ижод қилиш соҳасида;
9. Арҳив иши соҳасида;
10. Сунъий интелект тизимлари яратиш соҳасида;
11. Виртуал реалликда (ҳаққонийликда).

2. Маълумотларни ёзиш, сақлаш ва ўқишда хотирадан унумли фойдаланиш ва сиқиш амалларидаги муаммолар таҳлили ва алгоритмларининг аҳамияти

Хотира - программа ва берилганларни ёзиб куйиш, сақлаш ва танлаб олиш учун мулжалланган курилма. Мультимедиа тизимида хотиранинг ўрни жуда ҳам мухим ҳисобланади. Хотира маълум бир чекланган ҳужайраларнинг сонидан ташкил қилинган бўлиб, ҳар бири узининг ягона (уникал) номерига яъни манзилига эга. Ҳужайрага кириш унинг манзилига мурожаат килиш билан амалга оширилади.

Оператив хотира (ОХ) - керакли информацияни (командаларни) бошқарув қурилма (БҚ)га ва берилган маълумотларни арифметик ва мантиқий қурилма (АМҚ)га сақлаб қўювчи функционал қурилма. Оператив хотира мультимедиа маълумотлари билан ишлаганда айниқса ўзининг имкониятли даражасини кўрсатади. Аудио, видео, матн, анимация ва тасвирлар билан ишлаганда буларни вақтинчалик оператив хотирага ёзиш ва ундан ўқиб олиш амаллар ичида кўп вақт талаб қиладиган жараён ҳисобланади. Шунинг учун оператив хотиранинг хажми унинг имкониятлилик даражасини белгилаб беради десак муболаға бўлмаган бўларди.

Компьютер ёрдамида хисобланиладиган мультимедия масалалар, мультимедия маълумотлари устида ҳисобланиладиган алгоритмнинг мураккаблигига, дастлабки мультимедия маълумотларининг берилганларнинг сонига ва шу кабиларга нисбатан хотирада турли микдордаги информацияни сақланишини талаб этади. Шунинг учун хотира керакли бўлган мультимедия маълумотларининг (аудио, видео, матн, анимация ва тасвирлар) хажмига тенг миқдордаги ахборотни сақлаши керак, яъни хотиранинг сиғими катта бўлиши лозим. Бошка томондан Компьютернинг бошқа қурилмаларининг тезкорлигига мувофиқ хотира ҳам керакли тезкорликга эга булиши талаб этилади.



1-расм. Хотира даража сатҳлари

Хотиранинг сиғими қанча куп бўлса шунчали унга кириш секинлашади, (1-расм) айниқса мультимедия маълумотлари билан

ишланаётганда. Чунки кириш вақти (яъни тезкорлиги) хотирадан маълумотни аввалом бор топиб танлаб ўқиш ёки унга маълумотни ёзиш вақтидан келиб чиқади. Шунинг учун компьютерда сиғими ва ишлаш тезкорлигидан фарқ қилинадиган бир катор хотира қурилмалари бўлади.

Хотира қурилмаси, кириш имкониятини вақти(с), сиғим (б, бит)

- Регистрлар, кэш-хотира, энг тезкор ва хажми кам;
- Оператив хотира: ўрта тезкор ва хажми нисбатан катта;
- Ташқи хотира: секин аммо хажми ўта катта.

Тезкор хотиранинг турлари ва имконятлари.

Регистрлар процессор архитектурасини асосини ташкил этади. Бўлиши шарт бўлган регистрлар тупламидан куйидагиларни келтириш мумкин: Берилганларнинг Регистрлари (БР) - амалларни бажарилишида оралиқ натижаларни вақтинча сақлаш учун хизмат килади.

Аккумулятор Регистри (АР) – ҳисоб жараёнида вақтинча сақлаш учун қўлланиладиган регистр (масалан, кўпайтириш командаси бажарилишида унда натижа шаклланилади).

Стек кўрсатувчи регистр (СКР) - стек билан ишлаш амалларида қўллани-лади. Стек шундай берилганларнинг тузилишики, унинг ишлаш тамоилида охирги кирган - биринчи бўлиб чиқади, охирги ёзилган қиймат биринчи бўлиб ўқилади. Хозирча стеклар асосан остпрограммаларни ташкил килинишида қўлланилади деб айтиб куйямиз.

Индексли, кўрсатувчи ва базис регистрлар - хотирадаги операндлар манзилларини хисоблаш ва сақлаш учун қўлланилади.

Регистр - щётчиклар программаларда циклик блокларини (қисмларини) ташкил қилиш учун қўлланилади.

Умумий мақсадларга мўлжалланган регистрлар (УМР) – кўп компьютерларда ташкил килинади ва ихтиёрий максадларда қўлланилиши мумкин. Программачи программа тузишида УМР регистрнинг аниқ қўлланилишини таърифлайди. Улар берилганларни вақтинча сақлаш учун аккумуляторлар, ҳамда индексли, базавий, кўрсатувчи регистрлар

сифатида ишлатилиши мумкин. Регистрларнинг сони ва улар оралиғидаги боғланишлар процессор-нинг мураккаблиги ва нархига деярли таъсир этади. Аммо, бошқа томондан, регистрларнинг микдорини кўплиги ва улар имкониятининг туплам бойлиги программалаштиришни соддалаштиради ва програм таъминотини эгилувчан-лигини оширади. Айтиб ўтилган регистлардан ташқари АМҚнинг таркибига программа буйича кириб булмайдиган.ички тизим регистрлар кириши мумкин. Улар командалар бажарилишида информацияни ички узатиш вактида қўлланилади.

Ички регистрлар ва кэш-хотира дастурчига бўйсинмайди, шунинг учун асосий ҳаракатлар оператив хотирада жойлаштириладиган маълумотни кесимли хажмини аниқ жамлаб олиш мухим масалага айланади. Тасвирлар кесим ёрдамида қайта ишланади, шунинг учун кесимлар хажмининг оптимал хажмини топиш зарур.

Маълумотларни сиқиш усуллари ва стандартлари. JPEG – график форматлардан бири ҳисобланиб, тасвирларни ва уларга тегишли айрим маълумотларни сақлаш учун ишлатилади. JPEG маълумотларини ўзида сақлаётган файл одатта кенгаймасига эга бўлади: **JPEG, .jfif, .jpg, .JPG** ёки **.JPE** кабилар мисол бщлади. Улар ичида . jpg формати кщпгина операцион тизимлар учун умумий ҳисобланади.

Қўлланилиш соҳаси. JPEG алгоритми тасвирнинг ёруғлиги ва рангининг ҳақиқийлигини сақлқгқн ҳолда силлиқ ўтиш билан тасвир маълумотларини сиқиш учун қўлланилади. JPEG рақамли тасвирларга ишлов беришга мўлжалланган бўлиб, бу алгоритмнинг асосий қўлланилиш сохаси бу интернет ҳисобланади. Интернет тармоғида тасвир маълумотларини тез ва йўқотишларсиз узатишда мақбул ечим ҳисобланади.

Бошқа томондан таҳлил қилинганда JPEG алгоритми чизмаларни, ёзувли ва белгили графикларни сиқа олмайди. Бунга асосий сабаб шундаки, юқорида санаб ўтилган тасвирларда қўшни пикселлар ёруғлигини кескин ўзгариши ва рангларнинг кескин фарқ қилиши

хисобланади. Бундай турдаги тасвирлар сиқилмаган ҳолда бутунлигича хотирада сақланади. Бу эса хотирадан ўқиш ва ёзиш амалларида тезликнинг нисбатан пасайиб кетишига сабаб бўлади.

Бу алгоритмдаги яна бир камчиликлардан бири бу тасвирларни кўп поғонали қайта ишлашда яхши натижа бермайди, шунга кўра поғонали қайта ишлаш давомида тасвирларни хотирада сақлаганда жуда қўпол бузулганлик юзага келиши мумкин.

Яна бир унинг қўллаб бўлмас томони шундаки, бу алгоритмни энг кичик йўқотишлар бўлмаслиги шарт бўлган тасвирларда қўллаб бўлмайди. Бунга мисол сифатида астраномик ва тиббиётда натижалардан ҳосил қилинган тасвирлардир. Бундай ҳолатларда тасвирларни сиқишнинг яна бир усулларидан бири ҳисобланган JPEG стандартининг Lossless JPEG ёки JPEG-LS алгоритмини тавсия қилиниши мумкин.

JPEG алгоритми ёрдамида сиқиш. Бу алгоритм орқали сиқилганда тасвирнинг RGB ранглар тўплами YCbCr га ўзлаштирилади. Шуни таъкидлаб ўтиш лозимки, JPEG стандарти айнан YCbCr дан фойдаланишни қатъий такидламайди, тасвирлдарни қайта ишлашни ва хотирага ёзишни бошқа методлари ҳам мавжуд.

Ранглар учун жавоб берадиган Cb ва Cr каналларни RGB -> YCbCr ўзгартиришидан кейин, сийраклаштириш (subsampling[3]) мумкин, қайсики 4 та пикселдан ташкил топган 2x2 ўлчамга эга квадрат матрицанинг ёркинлик Y каналининг қийматлари Cb ва Cr каналларнинг қийматларига яқинлаштирилади (яқинлаштириш шакли «4:2:0»[4]). Бу ҳолда 2x2 блокнинг 12 та қиймати (4 Y, 4 Cb ва 4 Cr) ўрнига атига 6 та қиймати (4 Y ва Cb ва Cr қийматларидан 1 тадан) ишлатилади. Агар сиқилган тасвирни тиклаш натижасида ҳосил бўлган тасвир сифати фойдаланувчини қаноатлантирмаса, унда тасвирни сиқиш йўналиши фақат бир томонлама амалга оширилиши мумкин — вертикал ҳолатда («4:4:0» чизмаси) ёки горизантал ҳолатда («4:4:2» чизмаси), яна бир усули умуман ҳеч қанақа ҳолатда («4:4:4»).

Кейин босқичда ёриқлик компоненти Y ва ранглар учун жавоб берадиган Cb ва Cr каналлари 8x8 пикселли блокларга бўлиб ташланади. Ҳар бир блок устида дискрет косинус ўзгартириш амаллари бажарилади. Дискрет косинус ўзгартиришидан ҳосил қилинган кайфисентлар квантланади ва Хаффман кодлар орқали кодланади.

Дискрет косинус ўзгартириш кайфисентини квантлаш учун қўлланиладиган матрица JPEG файлининг сарлавха қисмида сақланади. Хотирада JPEG файлига маълумотлар сақлаш жараёнида кайфисентлар шундай жойлаштириладики, юқори сифатли кайфисентлар сифати паст кайфисентларга нисбатан кучлироқ квантлаш даражасига силжитилади. Бу эса тасвирдаги кичик деталларни дағаллаштиришга олиб келади. Қанча юқори даражада сиқиш, кайфисентларни шунча кучлироқ даражада квантлаш деган маънони англатади.

Хотирада тасвирни сақлаганда JPEG файлида сифат қийматлари кўрсатилади, шартли бирликлар берилади, мисол учун , 1 дан 100 гача ёки 1 дан 10 гача. Энг катта қиймат асосан сифат даражасига тегишли бўлади.

Дискрет косинус ўзгартириш - ортоганал ўзгартиришлардан бири ҳисобланади. Косинус ўзгартириш бу ҳақиқий сонлардан ташкил топган vector учун қўлланилади. Бу усул маълумотларни йўқотишлар эвазига амалга ошадиган алгоритмлардан фойдаланган ҳолда қўлланилади, мисол сифатида MPEG ёки JPEG ни келтиришимиз мумкин. Бу ўзгартириш Фуре ўзгартириш билан чамбарчас боғлиқ ҳисобланади.

Математик томонидан ечилишини кўрсак, бунда векторни ўзгартириш матрицасига кўпайтмасини ташкил этади. Бунда ҳосил қилинган матрица векторга кўпайтрилганига қадар транспаренланади.

Ҳар хил ҳосил қилинган сигналлар турли хил ДКЎ турларига тегишли бўлади. Қуйида келтирилган матрица бошланғич 4 та формула келтирилган:

$$\text{DCT-1}_n = \left[\cos\left(kl\frac{\pi}{n-1}\right)\right]_{0\le k,l<n}$$

$$\text{DCT-2}_n = \left[\cos\left(k(l+\frac{1}{2})\frac{\pi}{n}\right)\right]_{0\le k,l<n}$$

$$\text{DCT-3}_n = \left[\cos\left((k+\tfrac{1}{2})l\tfrac{\pi}{n}\right)\right]_{0\le k,l<n}$$

$$\text{DCT-4}_n = \left[\cos\left((k+\tfrac{1}{2})(l+\tfrac{1}{2})\tfrac{\pi}{n}\right)\right]_{0\le k,l<n}$$

Бундан ташқари шуни ҳам келтириб ўтиш лозимки, дискрет косинус ўзгартириш тасвирларни сиқишда ҳам кенг қўлланилади. Бунда дискрет косинус ўзгартириш тасвирни турли хил амплитуда ва частота синусоид йиғиндисидан ташкил топади. Фуре дискрет ўзгартиришини кўрадиган бўлсак, бунда тасвирнинг ихтиёрий қисмларни катта бўлмаган кайфисентлар орқали ҳам ифодалашимиз мумкин. Бу унинг хусусияти асосан тасвирларни сиқиш учун кенг қўлланилади.

Икки ўлчамлли дискрет косинус ўзгартириш MxN ўлчамли A матрицани қуйидаги формула орқали ифодалашимиз мумкин:

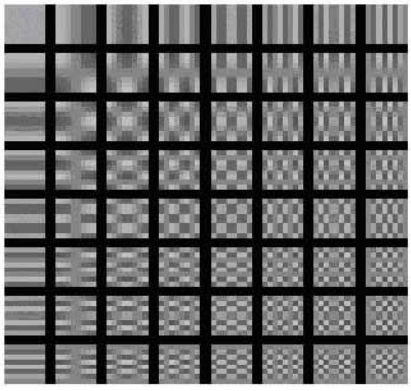
$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\frac{\pi(2m+1)p}{2M} \cos\frac{\pi(2n+1)q}{2N},$$

$0 \le p \le M-1$ и $0 \le q \le N-1$;

$$\alpha_p = \begin{cases} \dfrac{1}{\sqrt{M}}, & \text{агар} \quad p = 0; \\ \sqrt{\dfrac{2}{M}}, & \text{агар} \quad 1 \le p \le M-1. \end{cases} \qquad \alpha_q = \begin{cases} \dfrac{1}{\sqrt{N}}, & \text{агар} \quad q = 0; \\ \sqrt{\dfrac{2}{N}}, & \text{агар} \quad 1 \le q \le N-1. \end{cases}$$

B_{pq} - А матрирсанинг кайфиценти деб аталади.

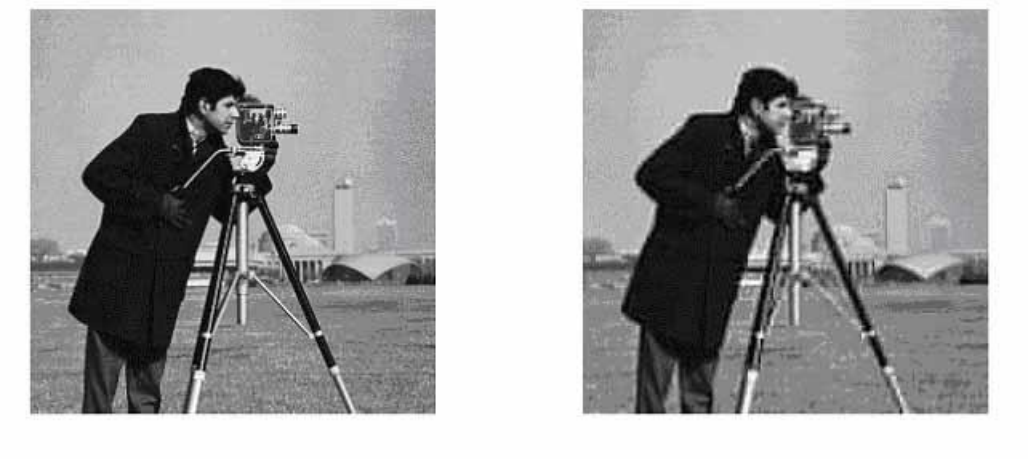
Мисол учун 8x8 матрица элементлари учун 64 базавий функцияси ишлатилади, қуйида келтирилган тасвирда намойиш этилган:



2-расм. 8x8 матрица элементлари учун 64 базавий функцияси

Дискрет косинус ўзгартириш ёрдамида тасвирларни сиқиш.

Биз қуйидаги тасвир мисолидан 8x8 ўлчамли матрица элементларига мўлжалланган 64 базали кайфисентни ишлатамиз. Бунда тасвирнинг ҳар бир блокидан 10 та элементни қолдирамиз ва қолган элемнтларни 0 қиймат билан алмаштирамиз. Демак 6 та элементдан 1 таси қолади. 3-расмда дискрет косинус ўзгартириш ёрдамида тасвирни сиқишдан олдинги ва кейинги кўринишлари келтирилган. Бунда асосий фарқи сифат жихатдан қайта ишланган қайта ишланган тасвир азгина пастроқ бўлади, аммо хажм жихатдан хотирадан кам жой эгаллайди.



3-расм. Тасвирда уларнинг фарқлари

3. Тасвирларни қайта ишлашда Вейвлет - Добеши ўзгартиришидан фойдаланиш

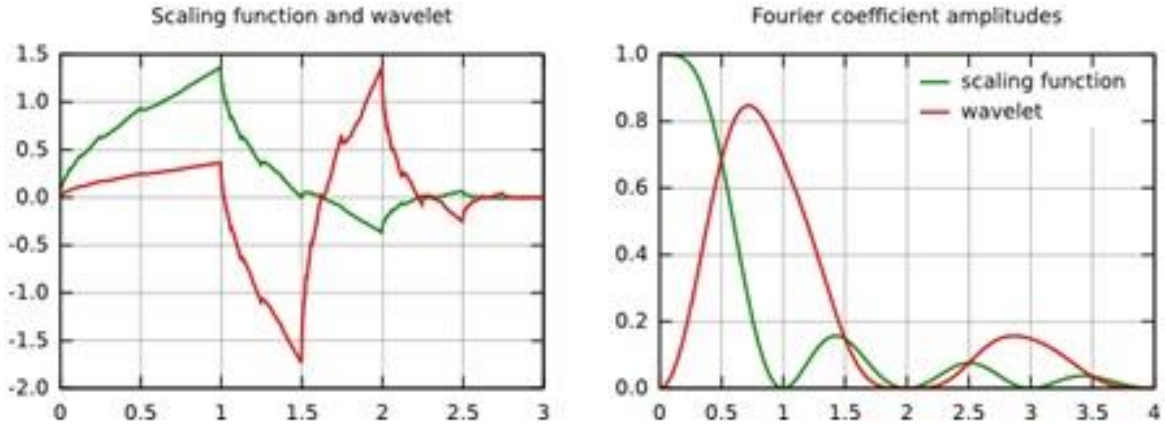
Вейвлет сўзи инглизча “wavlet” сўзидан олинган бўлиб, “ёйиш” деган маънони англатади. Вейвлетлар назарияси Фуре анализига мос алтернатив бўлиб, сигналларга ишлов берувчи янгича техника назариясини ташкил этади.

Вейвлетнинг афвзалликларидан бири, у сигналларни яхши махаллийлашган ўзгаришларни алмаштиришга имкон беради.

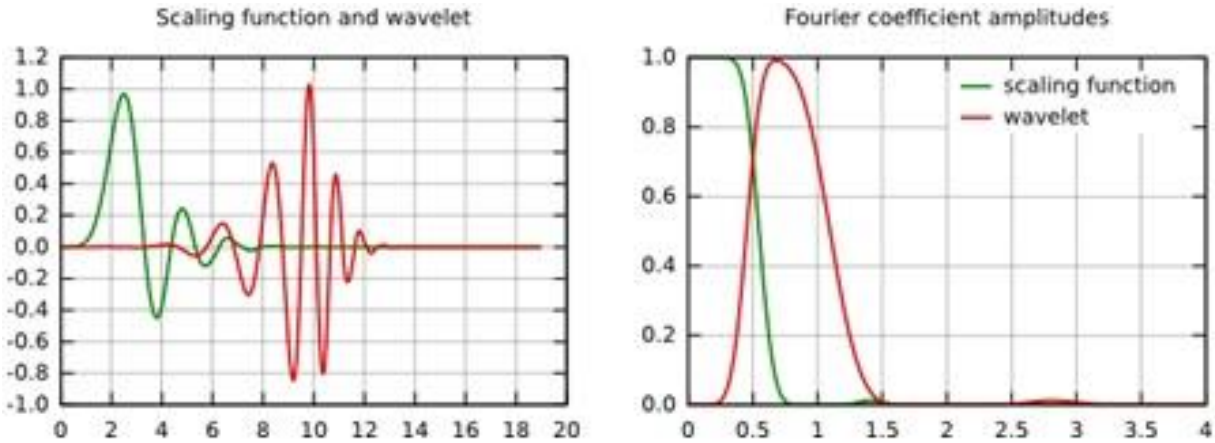
Фуре анализи эса бунга имкон бермайди. Фуре коэффициентларига сигнал мавжуд бўлиши барча вақтларидаги сигналнинг ҳолати акс этади. Сонли ва функсионал анализда дискрет вейвлет ўзгартириш Вейвлет ўзгартиришига мансуб бўлиб, уларда вейвлетлар дискрет сигналлар бўлиб ифодаланади.

Қуйидаги кўрсатилган график тасвирларда вейвлетлар ва маштабланган функциялар келтирилган. Айрим мукаммал қўлланиладиган синфлар ичида Добиши синфига мансуб вейвлетлар кўп қўлланилади.

Аналог икки ўлчовли сигнални вақт бўйича дискретлаш ва даража бўйича квантлаш натижасида рақамли тасвир (РТ) пайдо бўлади.



4- расм. Добеши 4 ва вейвлетнинг маштабланган функсиясининг частотали ташкил этувчилари



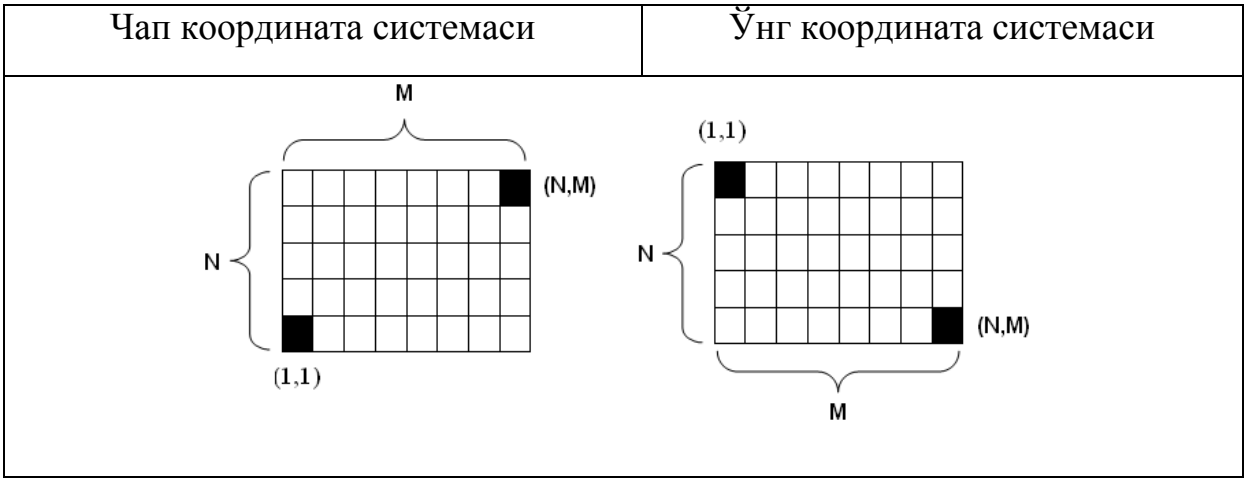
5 - расм. Добеши 20 ва вейвлетнинг маштабланган функсиясининг частотали ташкил этувчилари

РТнинг энг кичик элементи пиксел (пихел) деб аталади. РТ умумий ҳолда N та қатор ва M та устундан иборат тўғри бурчакли жадвал кўринишида берилади, бунда ҳар бир элемент пиксел бўлади. Бу жадвални NxM элементлардан иборат матриса кўринишида ҳам ёзиш мумкин.

РТ пикселларини координаталарини график тасвирлаш учун турли усуллардан фойдаланилади [9-10].

Тасвирларни таниб олиш масалаларида битта РТ турли усулларда келтирилиши мумкин, яъни декарт ёки қутбли координата системаларида.

6-расмда РТни икки хил усулда декарт координата системасида тасвирлаш кўрсатилган.



6-расм. Декарт координат системасида РТни икки хил усулда тасвирлаш

Чап координат система X ўқини чапдан ўнгга ёналишига мос келади. Ўнг координат система Y ўқини пастдан юқорига ёналишига мос келади. Шу сабабли РТни ифодаловчи матрисанинг пастки чап томонида (1,1) координатли пиксел жойлашади, юқори ўнг томонда эса (N,M) координатли пиксел жойлашади.

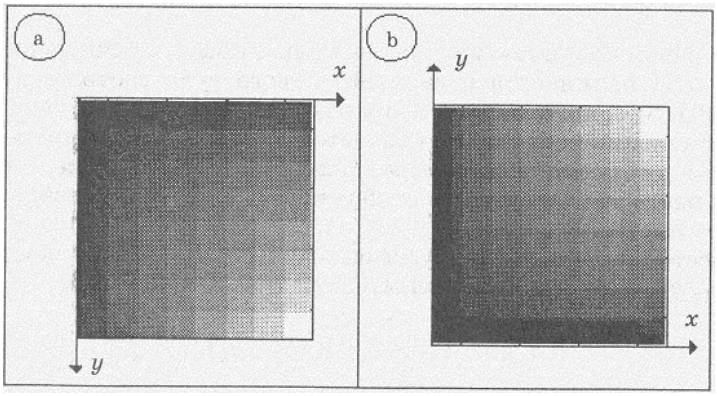
Ўнг координат системада РТ пикселларини тартибли ҳисоби унга мос матрисанинг юқори чап бурчагидан бошланиб ўнг пастки бурчакда тамомланади. Координатларнинг бундай ифодаланиши умум қабул қилинган икки ўлчовли чап декарт системага мос келмасада, у РТ XY

текисликда акс эттиришда кўп қўлланилади. (x1,y1) ва (x2,y2) координатали икки пиксел орасидаги д масофа қуйидагича аниқланади:

$$d_{1,2} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\,. \qquad (1)$$

Бизга 8x8 пиксел ўлчовли тасвирни аниқловчи 8-тартибли матрица берилган. Бу тасвирни декарт координат системасидаги график кўриниши 6-расмда кўрсатилган. Бу ерда а харфи билан (2) тасвирнинг чап координат системасидаги кўриниши, б харфи билан унинг ўнг координат системадаги кўриниши белгиланган.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \\ 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 \\ 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 \\ 6 & 12 & 18 & 24 & 30 & 36 & 42 & 48 \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 & 56 \\ 8 & 15 & 24 & 32 & 40 & 48 & 56 & 64 \end{bmatrix}, \qquad (2)$$

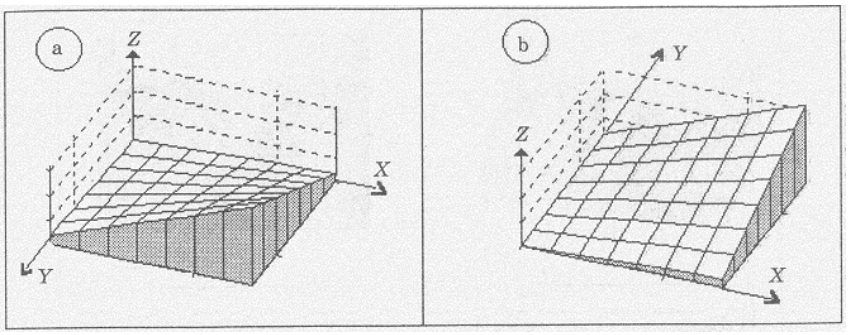


7-расм. (2)- матрисани рақамли тасвири

(2) матриса уч ўлчовли декарт координат системасида ҳам график кўринишда келтирилиши мумкин. Бу ҳолда матрисанинг элементлари XY текисликда жойлашади. Бу элементларнинг қийматлари Z ўқи бўйича қўйилади. Бундай тасаввурнинг натижаси 6-расмда кўрсатилган.

7-расмда а харф билан (2) тасвир чап уч ўлчовли декарт координат системасида белгиланган, б харф билан эса ўнг уч ўлчовли декарт координат системасида белгиланган.

[10] терминологияси бўйича 3Д кўринишида келтирилган тасвирлар “0” синф рақамли тасвирларга киради. «0 синфини» тасвирларни аниқлашда умум қабул қилинган синф тушунчаси билан адаштирмаслик учун уни усул деган тушунча билан алмаштирамиз.



8-расм. (2) матрисани 3Д тасвир шаклида ифодаланиши

[10] да рақамли тасвирларни таърифлаш ва ифодалаш учун бешта усул киритилган, улардан 1-4 усуллар тасвирларни 2Д шаклда ифодалашга мўлжалланган. Охирги усул ўзининг алоҳида нуқталари ёки локал соҳалари билан келтирилган яримтонли бинар, контурли ва тасвирларга бўлинган.

I боб бўйича хулоса

Предмет соҳасини таҳлил қилганимизда қуйидаги хулосаларга келиш мумкин:

1. Мультимедиа тизимларида ахборотни жойлаштириш ва саралаш амалларида хотира билан боғлиқ муаммоларни бартараф этиш йўллари мавжуд эканлиги ва буни ечиш йўли сифатида маълумотларни сиқиш алгоритмларини қўллаш.

2. Сиқиш алгоритмларини янгиттан яратиш эмас, балки анъанавий вейвлет-жараёнларини қўллаган ҳолда тасвирларга рақамли қайта ишлов бериш усулларини оптималлаштириш мумкинлиги.

3. Тасвир маълумотларини хотирадан ўқиш ва унда вақтинча сақлаш амалларини бажаришда компьютер хотира ресурсларидан (регистр, кэш-хотира ва оператив хотира) иложи борича унумли фойдаланиш мультимедиа тизимларида мавжуд «хатоликлар»ни бартараф этиш мумкинлиги кўрсатади.

4. Замонавий компьютерларда тасвирлар кесимланган (бўлинган) ҳолатда қайта ишланади. Тасвирларни бўлиш пайтида кесимлар хажми катта рол ўйнайди, чунки кесимни катта ёки кичиклиги ёзиш ва ўқиш буйруқларини сонини белгилайди ва унумдорликка таъсир кўрсатади.

5.Ҳозирги даврга қадар маълумотларнинг хотира билан боғлиқ кўпгина муаммолари ҳал қилинганлигини юқоридаги бобда таҳлил қилинди. Аммо айнан мультимедиа тизимларида ахборотни жойлаштириш, кичик хажмда сақлаш ва тезроқ рақамли қайта ишлов бериш амаллари айрим йўналишларда ҳанузгача муаммоли томонлари мавжудлиги кўриб чиқилди. Бунда айниқса мультимедиа маълумотларининг турли хил форматга эга эканлиги ва хажм жихатидан хотирадан кўп жой эгалаши юқоридаги «муаммоли тўхталишлар»га сабаб бўлаётганлигини аниқланди.

Бу илмий ишнинг асосий мақсади мультимедиа тизимларидаги тасвирни хотира ва процессор билан боғлиқ қайта ишлаш амалларини оптимал усулларини яратишдан иборат. Бу мақсадга эришиш учун қуйидаги вазифаларни амалга ошириш шартлари қўйилди:

- тасвирларни рақамли қайта ишлашда вейвлет жараёнларидан фойдаланиш;
- тасвирларни хотира ресурсларида сақлашда сиқиш алгоритмларининг оптимал усулларини яратиш;
- парраллелаштириш жараёнида бўлинган тасвирни кесимларни хотирага ёзиш/ўқиш вақтлари назарда тутилиши лозим.

II боб. Мультимедиа тизимларида ахборотга ишлов беришда хотира ва процессор унумдорлигини оширишдаги муаммоларни бартараф этиш йўллари

Олдинги бўлимимизда мультимедиа тизимларидаги хотира билан ва тасвирларни рақамли қайта ишлаш тезлигига боғлиқ муаммоларни абстрак ҳолда таҳлил қилдик. Бу муаммоларни бартараф этиш учун бир қанча сиқиш усулларидан фойдаланишимиз ва процессор унумдорлигини оширишда ҳар хил алгоритмлардан фойдаланишимиз мумкинлигини таҳлил қилдик. Аммо биз таҳлил қилган усул ва алгоримтларни амалиётда синамасдан ва уларни бир-бири билан таққосламасдан қандай натижаларга эришимизни била олмаймиз.

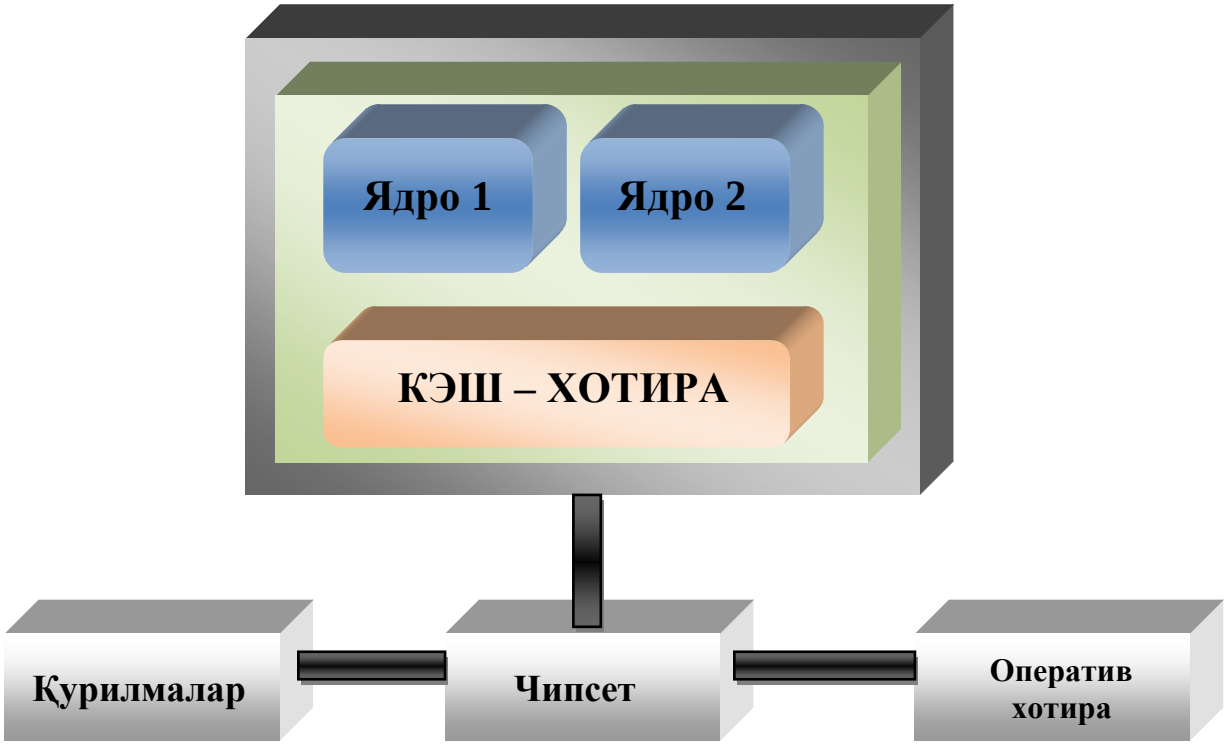
Шунинг учун, тасвирларни рақамли қайта ишлашда яхши натижага эришишнинг биринчи босқичида оптимал алгоритмларни танлаш каби масалалар ётади.

1. Мультимедиа тизимларида тасвирларни паралел қайта ишлаш алгортмлари ва техника воситалари

9-расм. Бир ядроли процессор архитектураси

Pentium микропроцессори 64 разрядли магистрал асосига курилган булиб хотира ва регистрлардан информация алмашинувини кескин тезлаштиради. Битта бажарувчи курилма иккита U ва V курилмалар билан алмаштирилган . Уларни хар бири паралел операциялар бажаради. U курилмаси асосида хисобланиб барча буйрукларни бажаради. V курилмаси ёрдамчи булиб факат энг куп учрайдиган буйрукларни бажаради. Ички регистр (кэш) иккига булинган: буйруклар кэши ва берилганлар кэши.

Pentium Prp да ички частотада ишловчи ички L2 кэш, хажми 256, 512 ёки 1024 уз контроллер ва локал 64-разрядли берилганлар шина кўшилган (10-расм). Қўшимча ички оптималлаштириш, тезлаштирлган конвейр ва паралеллаштириш даражаси ортирилган. Нисбатдан катта кувватли математик процессор урнатилган.

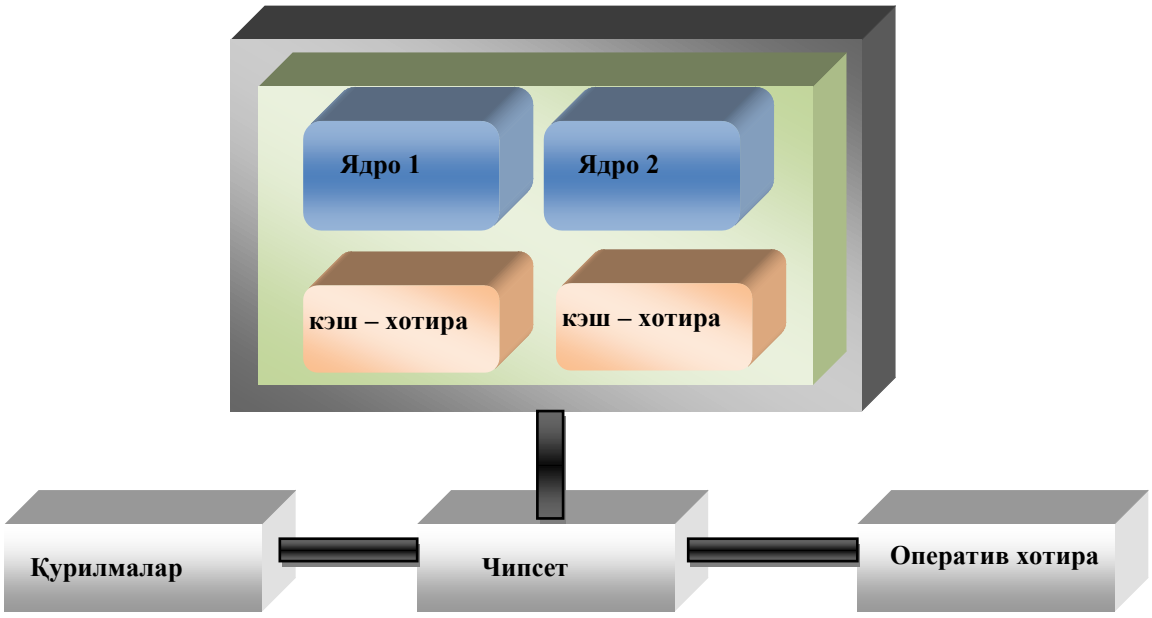


10-расм. Кўп ядроли процессорнинг архитектураси

Мультимедиа тизимларида тасвирлар устида ҳар хил амаллар кетма кетлигини бажаришда бир неча босқичлардан ўтилади [9,10]. Булар: мультимедиа ахборотини (мисол учун тасвир) хотирадан ўқиб олиш,

бажарилиш даражасига қараб процессор навбатига қўйиш, тасвир маълумотларини вақтинчалик сақлаш учун оператив хотирадан бўш жой ажратиш ва қайта ишлаш натижасида олинган тасвир қийматларини доимий хотирага ёзиш кабилардан иборатдир.

Аммо бу амаллар кетма кетлига ҳар доим ҳам аниқ вақт давомида якунланмайди ва вақтдан ютқазишга олиб келади. Бунга бир неча сабабларни мисол сифатида кўришимиз мумкин. Биринчидан, энг катта муаммолардан бири хотира муаммосидир. Тасвир хажмининг айнан бир хажмда белгиланмаганлиги, тасвирни ҳар қайта ишлаб, сақлаш учун хотирага мурожат қилинганда янги хажмда жой ажратишни талаб этади, бу эса тасвирга тегишли қийматларни ҳисоблаш, унинг кенгайтмасини аниқлаш ва бу ўзгаришларни операцион тизим журналида қайт этишни ва вақдан ютқазиш каби камчиликларга оли келади. Иккинчидан, қайта ишланадиган тасвир процессорга юклангач маълум чегараланган вақт ичида тасвир маълумотлари устида бажариладиган амаллар кетма кетлиги бажарилиб бўлиши лозим, акс ҳолда катта хажмдаги тасвир маълумотлари хотирага ёзиш давомида айрим кечикишларга олиб келиши мумкин.



11-расм. Кўп ядроли ажратилган кэш-хотирали процессор

Бундан ташқари шиналарни банд қилиш вақти хам чегараланган ҳисобланади. Компьютер хотирасида тасвирлар ҳажмининг ошиб кетиши қайта ишлаш тезлигининг пасайишига олиб келади. Мультимедиа тизимларида тасвирларни қайта ишлашда имкон қадар сифат даражасини пасайтирмаган ҳолда кичик ҳажмда сақлаш алгоритмларини ишлаб чиқишни талаб қилади. Тасвирларни қайта ишлашда кўп ядроли процессорлар учун мўлжалланган паралел қайта ишлаш алгоритмлардан фойдаланиш компьютернинг ишлаш унумдорлигига ижобий таъсирини кўрсатади. Аммо оқимларга ажратиш жараёнида тасвир маълумотларининг бир қайта ишлашда процессорга юкланадиган малумотлар ҳажми муҳим аҳамиятга эга. Чунки хотирадан вақтинчалик жой ажратиш масалалари ҳар доим дастур унумдорлигигни оширишдаги нозик жой ҳисобланади. Хотирадан жой ажратиш масалалрини амалга оширадиган параллелаштириш механизми қуйидаги масалаларни ечиши лозим:

1. оқим хафсизлигини таъминлаш;
2. қўшимча харажатлар;
3. мусобоқалаштиришни ташкиллаштириш;
4. хотира носозликлари.

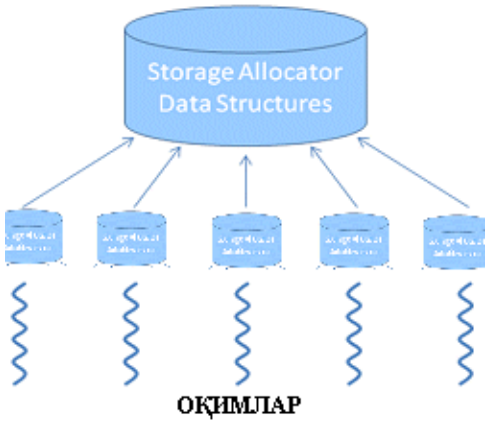
Икта оқим бир вақтнинг ўзида хотирани банд қилмоқчи ёки уни бўшатиши керак бўлиб қолганда, хотирани ажратиш механизмининг ички маълумотлар тузулишида мусобоқалаш бошланади ва дастурнинг нотўғри ишлашига олиб келади.

12 – расм. Оқимларнинг ажратилган хотирага мурожати

Агар оқимлар 12-расмда кўрсатилгандек тақсимланган хотирага чекланмаган мурожаат қилиш ҳуқуқига эга бўлсалар, унда оқимлар бир вақтнинг ўзида айнан бир ячейкага мурожаат қилишлари мумкин, бу эса тизим ишининг нормадан ташқари ишлашига олиб келади.

Муаммонинг оддий ечими сифатида хотирага мурожаат қилиш ва ундан вақтинчалик жой ажратишни чекловчи функцияларни ташкил этиш, бунда оқимларнинг хотирага мурожати бир амал бажариш давомида фақатгина бир марта мурожаат қили олиш ҳуқуқини беради. Натижада бир жараён бажарилиш давомида барча оқимлар хотирага бир маротаба мурожаат қилишади ва барча оқимлар бажарилиши синхрон тарзда якунланади.

Ажратилган хотирага мурожаат қилиш механизмига тақиқлашларни жорий этиш қуйидаги икта муаммони юзага келишига сабаб бўлиши мумкин. Биринчидан, хотирадан жой ажратиш ва ажратилган хотирани бўшатишга тақиқлашлар ўрнатиш оқибатида қўшимча кутилишларни секинлашишига олиб келиши мумкин. Иккинчидан эса, оқимларнинг хотирага рухсат олишидаги ҳаракати мусобоқалашиш жараёнларини юзага келтириши ва дастур ишини секинлашишига олиб келиши мумкин. Ҳар иккала муаммони 13-расмда кўрсатилгандек оқимларнинг ҳар бири учун хотирага локал мурожаатни ташкил этиш билан ечишимиз мумкин.



13 – расм. Ҳар бир оқим учун ажратилган локал хотира

13-расмда келтирилган технология ажратилган хотирани тақсимлаш деб аталади. Ажратилган хотирага оқимларни тақсимлаб бериш айниқса тасвмр мълумотларни параллел қайта ишлашда яхши натижа беради. Параллел қайта ишлашда тақсимланган тасвир маълумотларнинг квадрат матрицалар сонига ва уларнинг ўлчамига боғлиқ ҳисобланади. Матрицалар сонининг кўпайиши хотирага бўлган мурожатлар сонини шунчалик ошишига сабаб бўлади. Чунки матрицанинг ўламига лойиқ ажратилган хотирадан динамик жой тақсимлаш лозим бўлади, бу жараён эса маълум миқдорда вақт сарфланишига олиб келади. Тасвирларни рақамли қайта ишлаганда уларни ҳам процессор унумдорлигини ошириш муаммоларини ҳам хотира билан боғлиқ оқимларни тақсимлашдаги хатоликларни бартараф этишда оптимал усуллар яратилди. Уларни бобнинг давомида амалиётда қўлланилганмисоллар тариқаси

Ҳозирга пайтда мултимедиа маълумотининг бир тури ҳисобланган тасвирни сиқишнинг бир неча алгоритмлари мавжуд, булар ичида JPEG алгоритмини мисол қилишимиз мумкин. JPEG алгоритмининг йўқотишли ва йўқотишсиз сиқиш усуллари мавжуд. Йўқотишли усулда сиқишда тасвир ҳажми сезиларли камаймайди. Тасвир сифат даражаси муҳим аҳамиятга эга бўлмаган ҳолларда бу усулдан фойдаланиш яхши натижа бермайди ва мултимедиа тизимларида фойдаланганда хотира ҳажмидан ортиқча жой эгаллайди. Бу муаммони ечишда кўп ядроли процессорлар учун паралеллаштириш алгоритмларини қўллаган ҳолда тасвирларни йўқотишли сиқишнинг яна бир янги алгоритми ишлаб чиқилди [16].

Бу алгоритмда тасвирни икки ўлчамли Вейвлет - Добеши ўзгартириши қўлланилди. Вейвлет - Добеши дискрет ўзгартиришда тасвирни сиқишгача бўлган амаллар кетма-кетлигини тезлаштириш мақсадида C# дастурлаш тилининг паралеллаштириш кутубхоналаридан фойдаланилди. Бунда кўп ядроли процессорларда дастур алгоритмида бажариладиган арифметик амаллар бир вақтда ҳар бир ядрога тақсимлаб бўлиб берилди. Шундан сўнг Вейвлет – Добеши дискрет ўзгартиришдан

олинган массив қийматлари устида сиқиш амалларини паралеллаштириш алгоритмларидан фойдаланган ҳолда амалга оширилди.

Юқорида мисол сифатида келтирилган сабаблардан процессор билан боғлиқ томонларини оладиган бўлсак бунда асосан параллелаштириш амалларига тўхталсак муаммони ечимига яқинлашган бўлардик. Параллелаштиришнинг дастурий воситаларини оладиган бўлсак, энг аввало параллелаштириш технологиялари тўғрисида тушунчага эга бўлишимиз керак. Бунга OpenMP, TBB, Intel IPP, MPI [6,8,11,12] ва бошқа дастурий пакетлар мисол қилишимиз мумкин. Лекин булар ҳар бири қўлланилиш сохасига кўра фарқланади. OpenMP ҳақида батафсилроқ кўриб чиқамиз.

OpenMP технологиясининг хусусиятлари [12]. OpenMP(Open Multi - Processing) - кўп оқимли иловаларни яратиш учун мўлжалланган амалий дастурлашнинг интерфейси бўлиб, асосан умумий хотирага эга бўлган параллел ҳисоблаш тизимлари учун ишлаб чиқилган. OpenMP компиляторлар ва махсус функциялар библиотекаси учун директивалар тўпламидан иборат. OpenMP стандарти яқин 15 йил ичида умумий хотирага эга архитектураларга қўлланилган ҳолда яратилган.Сўнгги йилларда тақсимланган хотирали параллел ҳисоблаш тизимлари учун OpenMP стандартининг кенгайтирилган ҳолда ишлаб чиқилмоқда. 2005 - йилнинг охирида Intel компанияси Cluster OpenMP маҳсулотини тақдим этди, унда кенгайтирилган OpenMP ишлаб чиқилган бўлиб тақсимланган хотирали параллел ҳисоблаш тизимлари учун мўлжалланган.

OpenMP спецификациясини ҳисоблаш ва дастурлаш техникаси бўйича бир нечта йирик ишлаб чиқарувчи компаниялар (Intel, Hewlett-Packard, Silicon Graphics, Сун, IBM, Fujitsu, Hitachi, Siemens, Bull) яратишмоқда, уларни OpenMP Architecture Review Board(ARB) деб номланган нотижорат корхонаси томонидан бошқарилади.

OpenMP кўп оқимли иловаларни тез ва енгил яратишни Фортран ва C/C++ алгоритмик тилларда амалга оширади. OpenMP нинг биринчи

версияси 1997 - йилда Фортран тили учун яратилган. C/C++ дастурлаш тиллари учун эса 1998 - йилда яратилган. 2008 - йилда эса OpenMP нинг 3.0 версияси тақдим этилди.

OpenMP да параллел ва кетма – кетлик [11,12]. Параллел муҳитга кирилгандан сўнг янги `OMP_NUM_THREADS`-1 оқимлар яратилади, ҳар бир оқим ўзининг уникал номерига эга бўлади, бунда дастлабки оқим 0 номер билан белгиаланади ва у бош оқим (*master*) бўлади. Қолган оқимлар рақам сифатида бутун сонлар 1 дан `OMP_NUM_THREADS` - 1гача бўлади. Оқимлар сони белгиланган параллел муҳитда бажарилади ва ушбу муҳитдан чиқиб кетишгача ўзгармай қолади. Параллел муҳитдан чиқиб кетгандан сўнг синхронизация ёрдамида бош оқимдан бошқа барча оқимлар йўқ қилинади.

Қуйидаги мисолда параллел директиваси ишлаши келтирилган. Натижада бош оқим “1 - кетма - кет муҳит” матнини экранга чоп этади, кейинчалик параллел директиваси янги оқимларни ҳосил қилади ва ушбу оқимларнинг ҳар бири “параллел муҳит” матнини экранда чоп этади, кейин яратилган оқимлар тугатилади ва бош оқим “2 - кетма - кет муҳит” матнини экранга чоп этади.

```
#include "stdafx.h"
#include <omp.h>
using namespace System;
int main(array<System::String ^> ^args)
    {
         Console::WriteLine(“1 – ketma – ket  muhit”);
    #pragma omp parallel
         {
             Console::WriteLine(“parallel  muhit”);
         }
         Console::WriteLine(“2 – ketma – ket  muhit”);
    }
```

Айрим ҳолларда тизимнинг ўзи параллел муҳитда бажарилаётган оқимлар сонини тизим ресурсларини оптимизация қилиш учун динамик равишда ўзгартириши мумкин.Оқимлар сонини динамик равишда ўзгартириш `OMP_DYNAMIC` o’zgaruvchisiga `true` қийматни бериш орқали

амалга оширилади. Масалан, Linux операцион тизимининг баш команда оболочкасида ушбу қийматни қуйидаги буйруқ орқали амалга оширилиш мумкин:

```
export OMP_DYNAMIC = true;
```

Динамик равишда ўзгарадиган тизимларда оқимлар сони одатда белгиланмаган бўлади ва унинг қиймати фалсега тенг бўлади.

`omp_in_parallel()` функсияси 1 қийматни қайтаради, агар актив ҳолатдаги параллел муҳитдан чақирилган бўлса.

Қуйидаги мисолда `omp_in_parallel()` функсияси қўлланилган. модефунксияси қайси муҳитдан чақирилишига қараб, “параллел муҳит” ёки “кетма - кет муҳит” қаторларини чоп этишда қўлланилади.

```
#include "stdafx.h"
#include <omp.h>
using namespace System;
void mode(void)
{
     if(omp_in_parallel())
          Console::WriteLine("parallel muhit");
     else
          Console::WriteLine("ketma - ket muhit");
}
int main(array<System::String ^> ^args)
{
     mode();
#pragma omp parallel
     {
#pragma omp master
          {
               mode();
          }
     }
    return 0; }
```

C/C++ дастурлаш тилларида [22,25,29] юқоридаги барча шартлар *single* директиваси билан биргаликда эълон қилинади.

Дастурнинг белгиланган қисмини қайси оқим бажариши тавсифланмайди. Агарда nowait шарти эълон қилинмаса, битта оқим

белгиланган фрагментни бажаради, қолган оқимлар унинг ишини тугашини кутиб туради. single директиваси умумий ўзгарувчилар билан ишлаганда керак.

`Master` директивасикоднинг маълум бир қисмини фақат бош оқим бажариши учун белгилайди. Қолган оқимлар ушбу қисмни ўтказиб юборишади ва ундан қуйида турган оператор билан дастурни ишлашини давом эттиради. Ушбу директивада синхронизация амалга оширилмайди. C/C++ дастурлаш тилида директива қуйидагича эълон қилинади:

```
#pragma omp master
```

Қуйидаги мисолда мастердирективасининг ишлаши келтирилган. n ўзгарувчи локал ҳисобланиб, ҳар бир оқим ўзининг нусхалари билан ишлайди. Даставвал барча оқимлар n ўзгарувчига 1 қийматини ўзлаштиришади. Сўнгра бош оқим n ўзгарувчига 2 қийматини ўзлаштиради ва барча оқимлар ушбу қийматни экранга чоп этади. Мисолда кўриниб турибдики, мастердирективасини ҳар доим битта оқим бажаради. Ушбу мисолда барча оқимлар 1 сонини экранга чиқарса, бош оқим дастлаб 2 сонини, сўнгра эса 3 сонини экранга чоп этади:

```
#include "stdafx.h"
#include <omp.h>
using namespace System;
int main(array<System::String ^> ^args)
{
     int n;
#pragma omp parallel private(n)
     {
          n=1;
#pragma omp master
          {
               n=2;
          }
          Console::WriteLine("n ning birinchi qiymati: "+ n);
#pragma omp barrier
     #pragma omp master
          {
               n=3;
          }
          Console::WriteLine("n ning keyingi qiymati: "+ n);
     }
    return 0;
```

}

Қуйидаги мисолда `private` шартини ишлаши келтирилган. Ушбу мисолда параллел муҳитда n ўзгарувчи локал ўзгарувчи сифатида эълон қилинган. Бу ҳар бир оқимнинг n нинг нусхлари билан ишлашини билдиради ва ҳар бир оқимнинг бошида `n` ўзгарувчи инициализация қилинади. Дастурнинг бажарилиш вақтида `n` ўзгарувчининг қиймати тўртта турли хил жойларда чоп этилади. Биринчи марта n ўзгарувчининг қиймати 1 га ўзлаштирилгандан кейин кетма - кет муҳитда чоп этилади, иккинчи марта барча оқимлар n ўзгарувчининг нусхасини параллел муҳитнинг бошида чоп этади. Кейин барча оқимлар ўзининг тартиб номерини `omp_get_thread_num()` функсияси ёрдамида олинган қийматини `n` га ўзлаштириб чоп этишади. Параллел муҳит тугагандан сўнг n ўзгарувчининг қиймати яна бир марта чоп этилади, бунда унинг қиймати 1 га тенг бўлади (параллел муҳит ишлаши давомида ўзгармаган):

```
#include "stdafx.h"
#include <omp.h>
using namespace System;
int main(array<System::String ^> ^args)
{
     int n;
     Console::WriteLine("ketma-ket muhitga kirishdagi n ning
qiymati: " + n);
#pragma omp parallel private(n)
     {
          Console::WriteLine("parallel muhitga kirishdagi n ning
qiymati: " + n);
          n=omp_get_num_threads();
          Console::WriteLine("parallel muhitdan chiqishdagi n ning
qiymati: " + n);
     }
     Console::WriteLine("ketma-ket muhitdan chiqishdagi n ning
qiymati: " + n);
    return 0;
}
```

C/C++ дастурлаш тилларида дастурнинг параллел муҳитда аниқланган статик ўзгарувчилар умумий (`shared`) ўзарувчи ҳисобланади. Динамик ажратилган хотира ҳам умумий ҳисобланади, аммо кўрсатгич ҳам умумий, ҳам локал бўлиши мумкин.

`Threadprivate` директиваси C/C++ дастурлаш тилларида умумий бўлган ўзгарувчиларни локал кўринишига ўтказиб бериши мумкин. Глобал обектларни локал ўзгарувчиларини тўғри ишлатилишда дастурнинг турли қисмларида бир хил оқимлар томонидан ишлатилишига ишонч ҳосил қилиш керак. Локал ўзгарувчиларга турли параллел муҳитлардан мурожаат қилинса, унда уларнинг қийматини сақлаб қолиш учун ҳажмли параллел муҳитлар бўлмаслиги керак, оқимлар сони иккала муҳитларда ҳам бир ҳил бўлиши керак ва `OMP_DYNAMIC` ўзгарувчисининг қиймати биринчи муҳит бошлангандан иккинчи муҳит бошланганча `false` деб ўрнатилган бўлиши керак. `Threadprivate` типида эълон қилинган ўзгарувчилар OpenMP директиваларининг `copyin, copyprivate, schedule, num_threads_if` шартларидан бошқа шартларида ишлатиб бўлмайди.

Қулфлар. OpenMP да синхронизация вариантларидан бири қулфлар (locks) механизмидан фойдаланиш мумкин. Қулфлар сифатида умумий бутун сонли ўзгарувчилар (ҳажми адресни сақлаш учун етарли бўлиши керак) ишлатилади.Ушбу ўзгарувчилар синхронизация примитивлари параметрлари каби ишлатилиши керак.

Қулфлар қуйидаги учта ҳолатда бўлиши мумкин: инициализация қилинмаган,блокланган ёки блокланмаган. Блокланмаган қулф айрим оқимлар томонидан эгалланган бўлиши мумкин.Ундан сўнг унинг ҳолатини блоклашга ўтади.Блокланмаган қулфни айнан ўша оқим озод қилиши мумкин, ундан сўнг қулф блокланмаган ҳолатига ўтади.

Иккта турдаги қулфлар мавжуд: оддий қулфлар ва мураккаб қулфлар. Мураккаб қулфлар битта оқим томонидан озод қилинишидан олдин кўп маротаба эгалланиши мумкин, оддий қулфлар эса фақат бир марта эгалланиши мумкин. Мураккаб қулфлар учун эгалланганлик коэффициэнти (nesting count) тушунчаси киритилади. Дастлаб унинг қиймати нолга тенг бўлади, ҳар бир эгалланганда унинг қиймати бирга ошади ва ҳар бир озод қилинганда бирга камаяди. Мураккаб қулфлар эгалланганлик коэффициэнти нолга тенг бўлса блокланмаган ҳисобланади.

Оддий ва мураккаб қулфларни инициализация учун C/C++ дастурлаш тилларида [25,29] қуйидаги функсиялар ишлатилади:

```
void omp_init_lock(omp_lock_t *lock);
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

C/C++ дастурлаш тилларида [22] қуйидагича эълон қилинади:

```
#pragma omp flush [(рўйхат)]
```

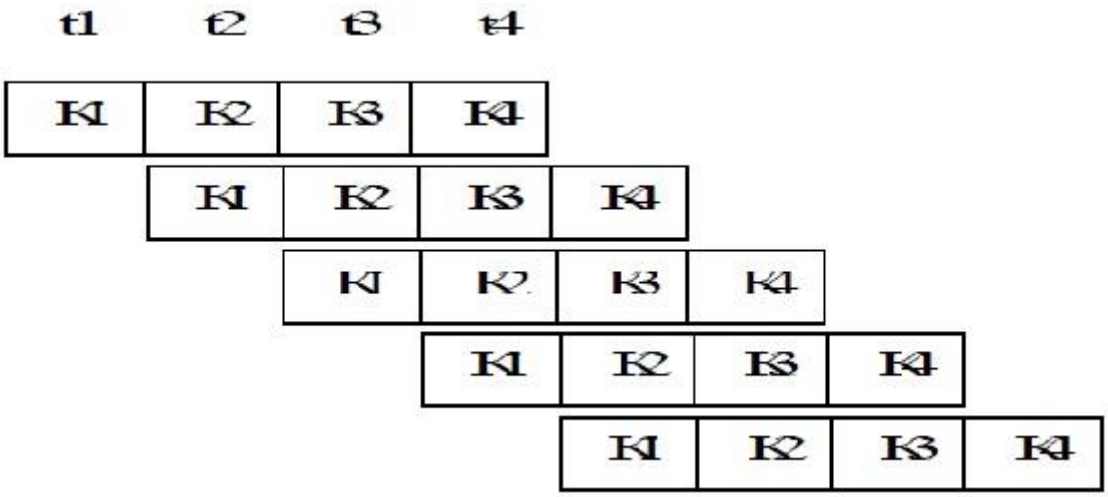
Ушбу директивани бажариш пайтида оқимнинг регистрларида ва кеш хотирасида сақланаётган ҳамма ўзгарувчиларнинг қиймати асосий хотирага киритилади; оқимнинг ишлаши давомида ҳамма ўзгарувчиларнинг ўзгариши қолган оқимлар учун кўринарли бўлади; агарда қандайдир ахборот чиқариш буферида сақланаётган бўлса, унда буферлар ташлаб юборилади. Бунда операция оқим томонидан чақирилган маълумотлар билан бажарилади, бошқа оқимлар томонидан ўзгартирилган маълумотлар тегинилмайди.

2. Бир ва кўп ядролик процессорларда параллелаштириш амалларининг ташкил этиш принциплари

Бир неча амалларни бир вақтда бажариш ғоясидан иборат бўлган маълумотларни параллел хисоблаш икки хил кўриниши мавжуд [13].

Булар: Параллел ва конвейер.

Агар бирор қурилма битта амални вақт бирлигида бажарса, у ҳолда мингта амални минг вақт бирлигида бажаради. Агар худди шундай бир вақтда ишлай оладиган ва бир–бирига мустақил бешта қурилма мавжуд деб қаралса, у ҳолда улар юқоридаги мингта амални мингта вақт бирлигида эмас, балки икки юзта вақт бирлигида бажаради. Худди шундай N та қурилмадан иборат тизим 1000 та амални 1000/N вақт бирлигида бажарида. Унга ўхшаш ҳолатларни ҳаётдан ҳам келтириш мумкин.



14-расм. Конвейерли қайта ишлаш архитектураси

Конвейерли қайта ишлаш [14]. Қўзғалувчан вергулли шаклда тасвирланган хақиқий иккита сонни қўшиш учун нима қилиш керак? Бунда бир қатор майда амаллар бажарилади. Булар: тартибини солиштириш, тартибини тенглаш, нормаллаш ва бошқа амаллар. Дастлабки компьютерларнинг процессорлари юқорида келтирилган барча “микро амалларни” ҳар бир аргументлар жуфтлиги учун натижани хосил қилгунча кетма-кет бажарган ва бундан кейин қўшилувчиларнинг кейинги жуфтлигини қайта ишлашга ўтган. Конвейерли қайта ишлаш ғоясида умумий амал бир неча босқичларга ажратилади. Ҳар бир босқич бажарилгандан кейин унинг натижаси кейинги босқичга берилади ва шу билан бирга кирувчи маълумотларнинг янги қисми қабул қилинади. Бунда олдин бажарилган амалларни натижаларини қўллаш орқали қайта ишлаш тезлиги оширилади. Фараз қилайлик, амал бешта микро амалдан иборат бўлиши ва уларни ҳар бири битта вақт бирлигида бажаради. Агар ажралмас ягона кетма-кет курилма мавжуд бўлса, у 100 та аргументлар жуфтлигини 500 вақт бирлигида бажаради. Агар ҳар бир микро амал конвейрли қурилманинг алоҳида босқичида бажарилса, у холда бундай қурилманинг ҳар бир қайта ишлаш босқичининг бешинчи вақт бирлигида биринчи 5та аргументлари аниқланади. Биринчи натижа вақтнинг 5-бирлигидан кейин олинади. 100 та жуфтликдан иборат тўплам эса 5+99=104 вақт бирлигидан

кейин олинади. Яъни параллел қурилмага нисбатан 5 марта тез бажарилади. Бир қарашда конвейерли қайта ишлашни параллел қурилмаларини ўрнига зарур миқдордаги конвейр қурилмаларини қўллаш мумкиндек кўринади. Бироқ бунда хосил бўлган система нархи ва мураккаблиги ошади. Унумдорлик эса ўзгармай қолади. Параллел дастур тузиш учун, дастурдаги бир вақтда ва бир-биридан мустақил процессорларда бажариладиган амаллар гуруҳини ажратиб олиш керак. Бунинг имконияти мавжудлиги дастурда информатцион боғлиқликлар мавжудлиги ёки йўқлиги билан аниқланади. Агар дастурнинг бирор амали натижаси иккинчи амал аргументи сифатида қўлланилса амаллар информацион боғлиқ деб аталади. Агар В амали А амалига информацион боғлиқ бўлса, у ҳолда В амали фақт А амали тугагандан кейин бажарилади. Агар А ва В амаллари информацион боғлиқмас бўлса, у холда алгоритмда уларни бажариш кетам-кетлигига чекланиш қўйилмайди, хусусан улар бир вақтда бажарилиши мумкун. Шундай қилиб, дастурни информацион боғлиқ амалларни аниқлашдан ва уларни хисоблаш қурилмаларига тақсимлашдан, синхронлашдан ва зарур коммуникацияни ўрнатишдан иборат бўлади.

Маълумотларни алмашиш ва қайта ишлаш усуллари. Массивли-параллел компьютерлар пайдо бўлиши билан параллел жараёнларни алоқасини қўлаб-қувватловчи кутубхона интерфейслар кенг тарқалди. Бу йўналишнинг тоифали вакилига Message Passing Interface (MPI) интерфейси мисол бўлади. Бу интерфейс амалда вектор-конвейерли супер - компьютердан тортиб шахсий компьютергача бўлган барча параллел платформаларда мавжуд. Дастурнинг қайси параллел жараёнлари дастурнинг қайси қисмида ва жараёнлар билан маълумотлар алмашиши ёки ўз ишини синхронлаб бориши кераклигини дастурчининг ўзи белгилайди. Одатда параллел жараёнларнинг манзил сохаси турлича бўлади. Хусусан, бу ғояга MPI да амал қилинади. Бошқа технологияларда, масалан Shmem да локал (private) ва умумий (shared) ўзгарувчилари қўлланилади. Бу

ўзгарувчиларга дастурнинг барча жараёнлари мурожат этиши мумкин ва Put/Get тоифасидаги операциялар ёрдамида умумий хотира билан ишлаш усули ташкил этилади. Linda системаси ўзига хос хусусиятга эга бўлиб, унда ихтиёрий кетма-кетликда тилга тўртта: in, out, read ва eval функцияларини қўшади ва параллел дастурлар тузиш имконини беради. Афсуски, ушбу келтирилган ғоя оддий бўлишига қарамасдан, уни амалда қўллаш муаммолар туғдиради.

Хабарларни алмашиш интерфейс технологияси. Тақсимланган хотира параллел компьютерлардаги кенг тарқалган дастурлаш технологияси MPI технологияси [15] хисобланади. Бундай системалардаги параллел жараёнларнинг ўзаро мулоқат усули бир-бири билан хабарлар алмашишдан иборат. Бу усул технологияси номи - Message Passing Interface (хабарлар алмашиш интерфейси) деб акс эттирилган. MPI стандартида система амал қилиши ва дастур яратишда фойдаланувчилар амал қилиши керак бўлган қоидалар мавжуд. MPI Фортран ва Си билан ишлашни қўллаб-қувватлайди. Интерфейснинг тўлиқ версиясида 125 тадан кўпроқ процедура ва функциялар мавжуд. MPI MIMD(Multiple Instruction Multiple Data) стилидаги параллел дастурларни қўллаб қувватлайди. Унда турли матнлар билан берилган жараёнлар бирлаштирилади. Бироқ бундай дастурларни тузиш ва отладка этиш мураккаб. Шунинг учун амалда дастурчилар SPMD (SINGLE PROGRAM MULTIPLE DATA) параллел дастурлаш моделидан фойдаланишади. Унда параллел жараёнлар учун битта дастур коди қўлланилади.

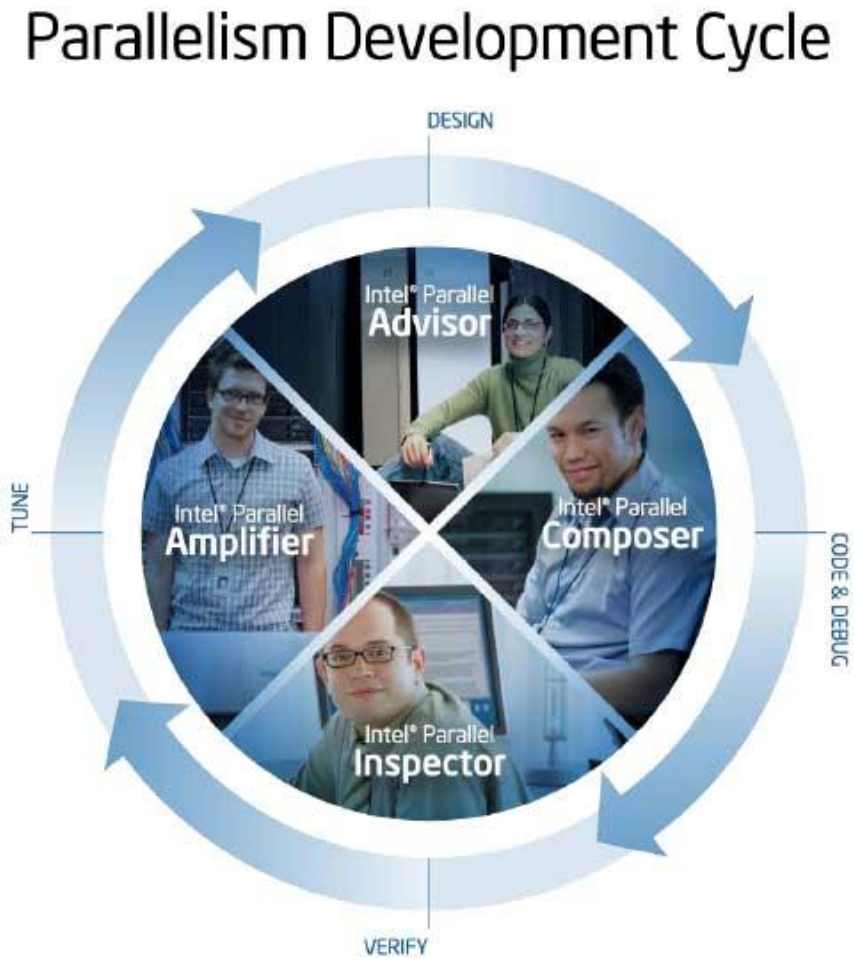
MPI технологияси[15]. Дстурни параллел кисмини ўрнатиш MPI_INIT(IERR). Қолган MPI працедуралар MPI_INIT чақирилгандан кейин чақирилиши мумкун. Хар бир дастурни параллел қисмини ўрнатиш фақат бир марта бажарилади Си тилида MPI_INIT функциясида кўрсатгичлар ёрдамида дастурнинг буйруқ сатридаги argc ва argv параметрлари берилади. Улар ёрдамида праралел жараёнларга параметрлар берилади.

3.Кўп ядролик процессорлар учун мўлжалланган дастурий пакетлар

Intel Parallel Studio да параллеллаштириш функсиясидан фойдаланиш [22]. Ҳозирги кунга келиб клиентли Windows иловаларни ишлаб чиқарувчи мутахасиссларга мавжуд ёки янги яратилаётган иловалар учун кучли ва қулай бўлган, фойдаланувчи тизимлари мултиядерли процессорларнинг базасида максимал равишда ишлай оладиган воситалар керак бўлмоқда.

Бугунги кунда энг кенг тарқалган Microsoft Visual Studio иловалар ишлаб чиқарувчи мутахасисснинг стандарт жамланмаси ҳисобланади. Intel компанияси Visual Studio нинг имкомиятларини кенгайтириш учун, яъни Windows учун яратилаётдан параллел дастурларни енгиллаштириш ва оптималлаштириш учун Visual Studio га plug - in сифатида Intel Parallel Studio ни ишлаб чиқди. Ҳозирги кунга келиб дастурларнинг унумдорлиги уларнинг қанчалик параллеллашгани ва тизимдаги процессорларнинг ошиши билан мутаносиб бўла олиши ҳисобга олинмоқда. Идеал дастур яқин йиллар ичида ядролар сони кўпайиб бораётган вақтда ишлаб чиқарилаётган янги процессорларнинг ҳамма қувватидан автоматик тарзда фойдаланади.

Intel Parallel Studio - бу бир нечта воситалардан иборат бўлган жамланма бўлиб, Microsoft Visual Studio га боғланган мултиядерли тизимлар учун унумдорлиги юқори бўлган параллел дастурлар ишлаб чиқарувчи восита ҳисобланади. Ушбу жамланма Visual Studio га plug - in бўлишидан ташқари, дастурчи томонидан ишлаб чиқилаётган дастурнинг ҳамма босқичида жумладан, дастурнинг скелетидан бошлаб охирги вариантининг оптимизациясигача қатнашади. Унинг таркибига тўртта алоҳида ҳар бири ишлаб чиқариш циклида қатнашадиган маҳсулот киради.



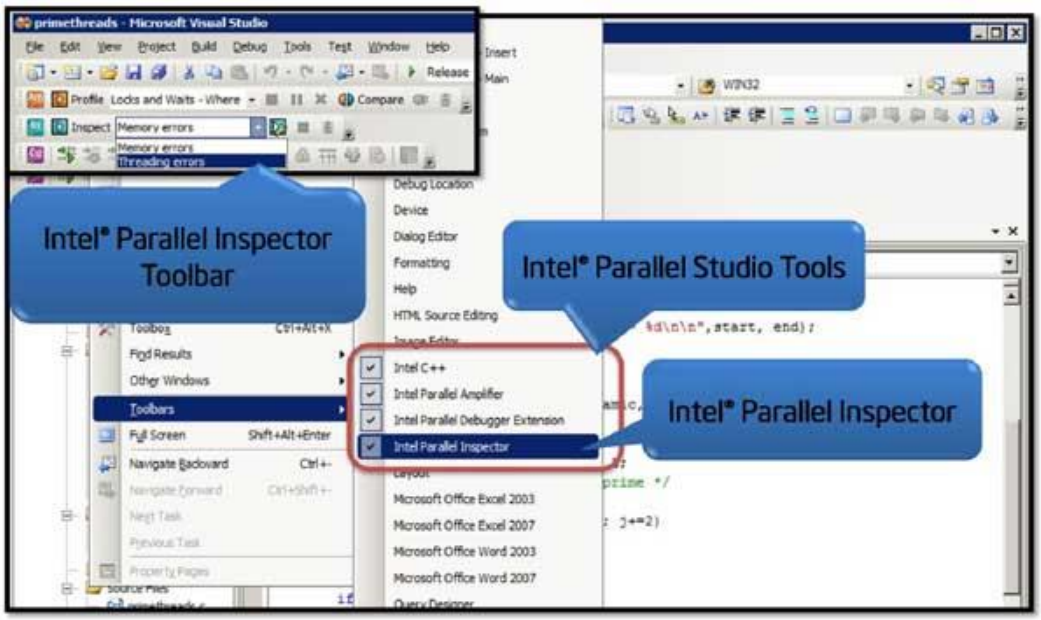
15- расм. Intel Parallel Studio ёрдамида яратиладиган дастурларнинг цикли

Пакет таркибига қуйидагилар киради:

- Intel Parallel Advisor - дастурнинг ишлаб чиқаришнинг бошидан кодни параллеллаштириш имкониятларини топишга ёрдам беради;
- Intel Parallel Composer - параллел код генерация қилиш учун мўлжалланган бўлиб, дастурни компилятор ва кўп оқимли алгоритмлар учун мўлжалланган кутубхоналар мажмусидан фойдаланилади;
- Intel Parallel Inspector - параллел дастурни тўғрилигини текширади ва хотира билан ишлаш хатоларини топиб беради;

- Intel Parallel Amplifier - дастурнинг мултиядерли платформаларда мутаносиблигини оширишга ҳалақит қилувчи “тор жойлари” ни аниқлайди.

Яратиш цикли узлуксиз ва қайтариладиган ҳолларда ушбу воситалар бир муҳитда уйғунлик билан ва бир - бирига ўзаро боғланган ҳолда ишлаши жуда муҳим ҳисобланади. Хатолар оқимини топиш ёки дастурни оптималлаштириш учун алоҳида профилировшикларни ишга тушириш ва бошқа эксперементлар қилиш талаб қилинмаслиги учун улар Microsoft Visual Studio муҳити билан яхлит бирлаштирилган. Ушбу воситалар Visual Studio нинг toolbarида ёки менюсида жойлашган бўлиб, анализ натижалари эса проект файллари турган жойда жойлаштирилади.



16 - расм. Visual Studio менюсида жойлашган
Intel Parallel Studio воситалари

Visual Studio муҳитида ушбу воситаларнинг қанчалик уйғунлашганлигини ёзиш қийин масаладир.Уларнинг функсионаллиги Visual Studio нинг давоми бўлиб, ажойиб параллел дастурлар тузишда фойдаланиш муҳим ҳисобланади.

Intel Parallel Advisor. Параллел дастур тузишнинг икки хил усули мавжуд.Биринчи усул мавжуд дастурни айрим изоляцияланган қисмларини, одатда проектнинг бутун архитектурасига тегинмайдиган алгоритмларни ишлаш тезлигини ошириш учун қисмли ёки бутунлай параллеллаштиришдир. Бунда дастурчи дастурни анализ қилади ва дастурда микропроцессорнинг максимал ресурсларини эгаллайдиган қисмларини белгилайди. Сўнгра проектнинг структураси анализ қилинади ва алгоритмни модификация қилиш қарори қабул қилинади. Параллел дастур тузишнинг иккинчи усули параллел бажариладиган юкланишларни ҳисоблаб бориш ҳисобланади.Агар проектни биргаликда бажарила оладиган қисмларга бўлиб бўлса, уларнинг дастур кўринишидаги реализацияси одатда қийин масала ҳисобланади.

Бу ерда ёрдамга Parallel Advisor келади. Бу мутлақо янги воситалар жамланмаси бўлиб, ўзида параллел дастурларни нолдан яратиш методологиясига ва уларнинг реализацияси учун тўғри қоидаларга эга. Parallel Advisor дастурлар параллел яратилганда эффектив бўлмаган ҳолатларини топади ва керакли ечимларни беради. Бундан ташқари параллел кутубхоналарни ишлатиш бўйича барча билимлар семпллар ва шаблонлар кўринишида йиғилган бўлиб, бошланғич босқичда улардан фойдаланиш учун максимал равишда енгиллаштирилган. Тайёр дастурни параллеллаштириш учун, илованинг тўғрилигини текшириш ва оптимизация қилиш учун Адвисор “ёъл кўрсатувчи оқим” ни ёки workflow ни тақдим қилади.

Intel Parallel Composer - IPP унумдорлик ва TBB параллел кутубхоналари билан Висуал Студиога боғланган бўлиб, параллел дастурлашни эндигина бошлаганлар ва ўзининг дастурини унумдорлигини Интел технологияси бўйича оширувчилар учун қулай ҳисобланади.

Composer қуйидаги компонентлардан тузилган:

- IPP кутубхонасида функсия сифатида яратилган ҳисоблаш примитивлари Интел платформаларидаги алгоритмларини юқори унумдорлигини кафолатлайди;
- Олдинги версияларда бўлмаган, лекин OpenMP 3.0 стандартнинг янги версияларида мавжуд бўлган мултитаскинг дан фойдаланади;
- Векторли операцияларни бажарувчи маълумотларнинг янги типи Валаррай кодни бир мунча енгиллаштиради, компилятор эса унумдорликни ошириш учун SIMD - инструксияларни ишлатиб, самарадор бинар кодни генерация қилади;
- Компийлатор томонидан C++ нинг стандарт элементларини қўллаб - қувватлайди.

Intel Parallel Inspector - ушбу восита бугунги кунда энг кўп талаб қилинган ва кутилган восита ҳисобланади, у кўп оқимли дастурнинг верификация босқичида бажарилиш турғунлигини ошириб хатоликлардан қутилиш учун ёрдам беради. Inspector фақатгина тестировшик командалар (QA team) билан ишлатилмайди.

Parallel Inspector қуйидагича ишлайди. Иккта турдаги хатоликлар классини топади ва улар бир - биридан алоҳида ишга туширилади: кўп оқимли хатоликлар ва хотира билан ишлаш хатоликлари. Хатоликларнинг охирги класси дастурчилар учун яхши маълум бўлиб, хотиранинг нотўғри ишлатилиши ва стекнинг бутунлигини бузилганлигини ёки мавжуд бўлмаган манзиллардан фойдаланишни аниқловчи турли воситалардан фойдаланишган. Хатоликларнинг иккинчи класси кўп оқимли дастурларнинг табиати билан боғлиқ. Улар параллел дастурларни яратиш пайтида ҳосил бўлади ва уларни аниқлаш қийин ҳисобланади.

Intel Parallel Amplifier - унумдорлик профилировшики бўлиб, илова томонидан мултипроцессорли платформа қанчалик самарадор ишлатилишини ва дастурнинг унумдорлигини оширишга ҳалақит берувчи нозик жойларни аниқлайди.

II боб бўйича хулоса

1. Бу бобда тасвирларни рақамли қайта ишлаганда параллелаштириш алгоритмларини қўллаш ва техник воситаларини асосий параметрларини белгилаб берилди.

2. Параллелаштириш алгоритмларидан фойдаланишда OpenMP кампилятори директивалари имконятларидан фойдаланиш усуллари кўрсатилди ва тасвирларни қайта ишлашда яхши натижа бера оладиган параллелаштиришнинг асосий OpenMP директивалири танлаб олинди.

3. Тасвирларни сиқиш дастурининг унумдорлиги унинг қанчалик параллеллашгани ва тизимдаги процессорларнинг ошиши билан мутаносиб бўла олиши ҳисобга олинди. Бунинг дастурий таъминоти сифатида Intel Parallel Studio дастурий жамланмадан фойдаланиш усуллари кўрсатилди.

4. Тасвирларни рақамли қайта ишлашда параллелаштиришнинг юқоридаги бобда келтирилган алгоритмлари кўп ядроли процессорларда яхши натижа берилиши ўрганиб чиқилди.

5. Ўзгарувчилар учун хотирадан вақтинчалик параллел тарзда жой ажратишда дастурий пакетлар хафсиз ва ажратилган оқимларни таъминлаб бериш тасдиқланди.

III боб. Тасвирларни қайта ишлашда қўлланилган паралеллаштириш алгоритмлари ва унумдорлик натижаларининг таҳлили

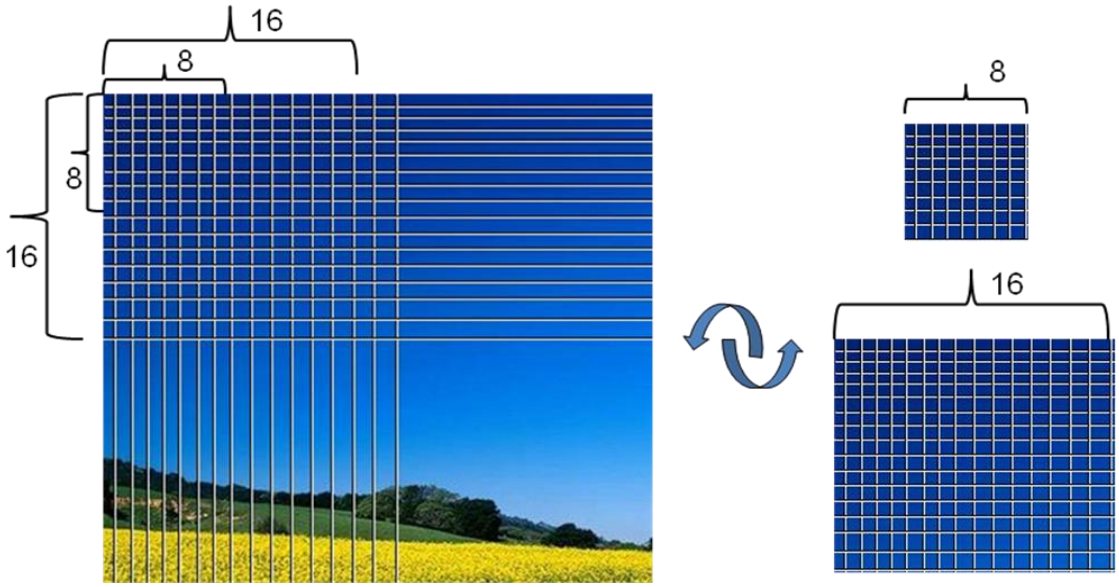
Бу бўлимда тасвирларни қайта ишлашда параллелаштириш методларининг дастурий алгоритмларда қўлланилишини ва уларнинг амалиётда кўп ядроли процессорларда тадбиғи, дастурий тил сифатида C++ [23] тилининг қўлланиши ва параллелаштиришни кўп оқимли қилиб ташкиллаштириб берадиган OpenMP комплятори директива имконятлари тўғрисига бағишланади. Бундан ташқари бу бўлимда дастурда қўлланилаган параллелаштириш алгоритмларининг кўп ядроли процессорларда қўллаш натижасида эришилган унумдорлик даражасининг, хотирадан ўқиш ва ёзиш амалларини оптималлаштириш мақсадида тасвирларни сиқиш муаммоларни ечишда дастурий воситалардан ва оқимларга ажратиш усулларидан фойдаланиш ҳақида айтиб ўтилади.

Бу муаммоларни ечиш мақсадида сиқиш амалларида Вейвлет-жараён орқали қайта ишлашни ва амаллар кетма кетлигини бажаришда паралеллаштириш алгоритмларидан фойдаланилди. Вейвлет-жараён орқали тасвир маълумотларини сиқиш – хозирга пайтда кенг қўлланиладиган сиқиш усулларидан бири ҳисобланади. Содда ҳолда тушунтириш мақсадида тасвирнинг ташкил топиш қисмлари ҳақида қисқача тўхталиб ўтилади.

1. Тасвирлани Вейвлет-жараён орқали қайта ишлашда паралеллаштириш алгоритмларининг унумдорлик даражаси

Тасвир — икки ўлчамли матрицадан ташкил топган маълумотлар тўплами бўлиб, матрицанинг ҳар бир ячейкаси яна учта қийматни ўзида сақлаган массивдан иборат. Агар биз оддий оқ-қора тасвирларни оладиган бўлсак, унда ҳар бир ячейкада ранг ўрнида қисм бўлакнинг ёркинлиги ҳақидаги маълумот сақланади, яъни 0 ва 1. Бу ҳолатда 0 қора ранг ҳақида

маълумот беради, 1 қиймати эса оқ ранг ҳақида маълумот беради. Тасвирнинг сифат даражаси ва хотирадан эгаллайдиган хажми матрицада иштирок этадиган қийматларнинг 0 дан 255 гача бўлган сонларнинг қатнашиш даражаси ва тасвирнинг ёрқинлик, аниқлик ва тиниқлик қиймат даражасига боғлиқ бўлади. Аммо тасвир ўлчами ва унинг кенгаймаси тасвир сифат даражасига ва хажмининг ошиб ёки камайиб кетишига ҳам тасир кўрсатади. Бунинг натижасида хотирадан меёридан ташқари кўп жой эгаллаши ёки сифатининг пасайиб кетишига сабаб бўлади.

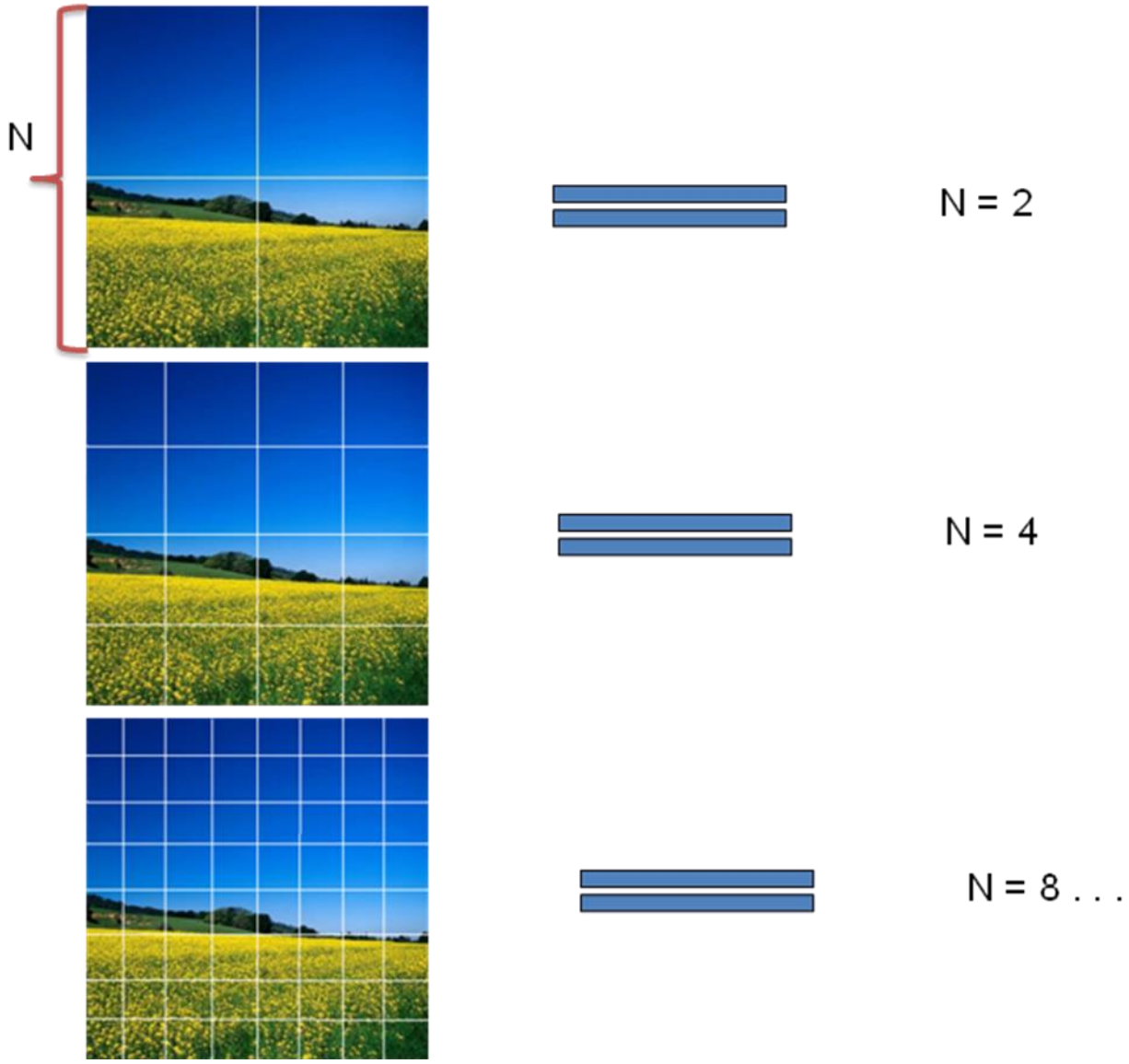


17 - расм. Тасвирни бўлмасдан туриб тасвир заррачаларга ажратиш

Ҳаттоки унча катта бўлмаган тасвир ҳам хотирадан кўп жой эгаллаши мумкин. Агар биз тасвирнинг ҳар бир пиксел ёрқинлигини бир байтдан қийматласак, унда оддийгина FullHD (1920×1080) форматининг бир кадри тахминан хотирадан икки мегабайни эгаллайди. Бунда умумий кетма-кетликдан ташкил топган мультимедиа ахбороти хотирадан жуда катта жой эгаллайди. Натижада мультимедиа маълумотларини хотирага жойлаштириш, сақлаш ва уни ўқиш каби амалларда айрим турдаги муаммоларни келиб чиқишига ва тизим ишига салбий таъсир кўрсатишига олиб келади.

Хотирани самарадорлигини ошириш учун:

- вейвлет ёрдамида сиқиш усуллари қўлланилади (19-расм);
- тасвирни бўлмасдан туриб тасвир заррачаларини оптимал хажмини қидириб топилади ;
- тасвирни умумий ҳолатини 2^n кесимларга бўлинади ҳар бир кесим алохида қайта ишланади.
-



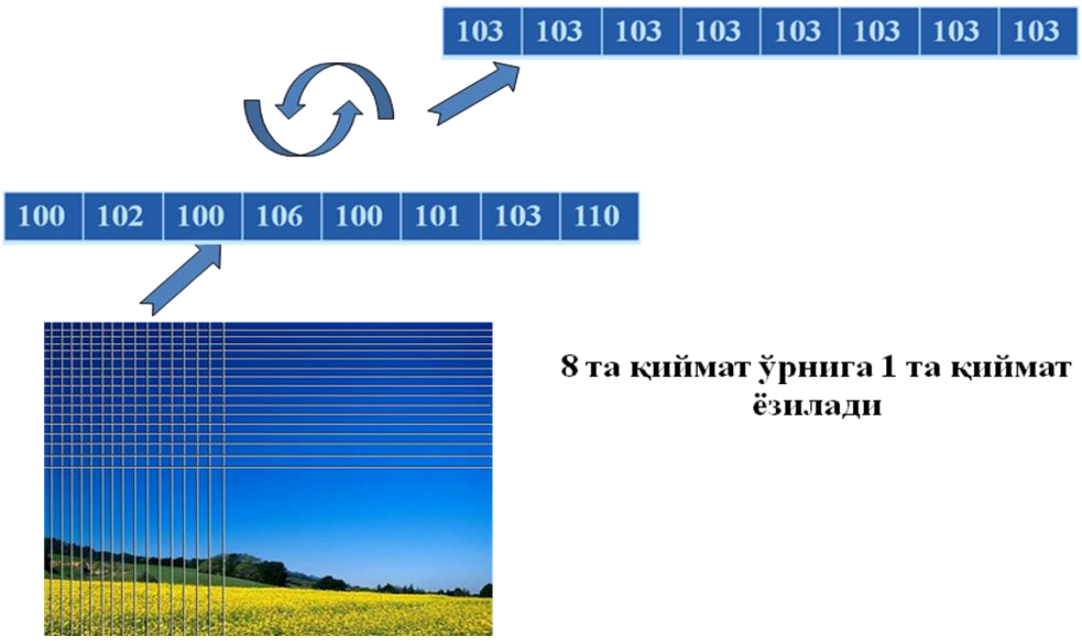
18-расм. Тасвирни умумий ҳолатини 2^N кесимларга бўлиш

Шунинг мақсадида тасвирларни сиқиш амалларининг оптимал йўллари яратилади. Тасвирларни қайта ишлаганда шундай амалларни

оптималлаштириш зарурки, бунда қайта ишлаш натижасида ҳосил бўлган тасвир маълумотлари хотирадан кам жой эгалласин. Тасвир фойдаланувчи учун компьютер экранида акслганда эса яна сифат даражасини тиклаган ҳолда намойиш этиши муҳим ҳсобланади.

Хозирги даврда сиқиш йўлларининг бер қанча усуллари мавжуд ҳисобланади. Уларнинг номларини тасвир кенгайтмасидан билиш мумкин. Булар ўзининг ютуқ томонлари бўлгани билан камчилик томонлари ҳам мавжуд. Шу сабабга кўра уларни қўлланилиш йўналишлари мавжуд.

Мисол сифатида, оддий реал хаётдаги тасвирни PNG кенгайтмада сақласак, хотирада эгаллаган хажми бизни хайратда қолдириши мумкин. Бунга асосий сабаб шундаки, ҳар бир тасвир бўлаги ёрқинлиги қўшни қисм ёрқинлиги билан камдан кам ҳолда бир хил қийматда бўлади. Бу эса сиқиш жараёнида яхши натижа бермайди.

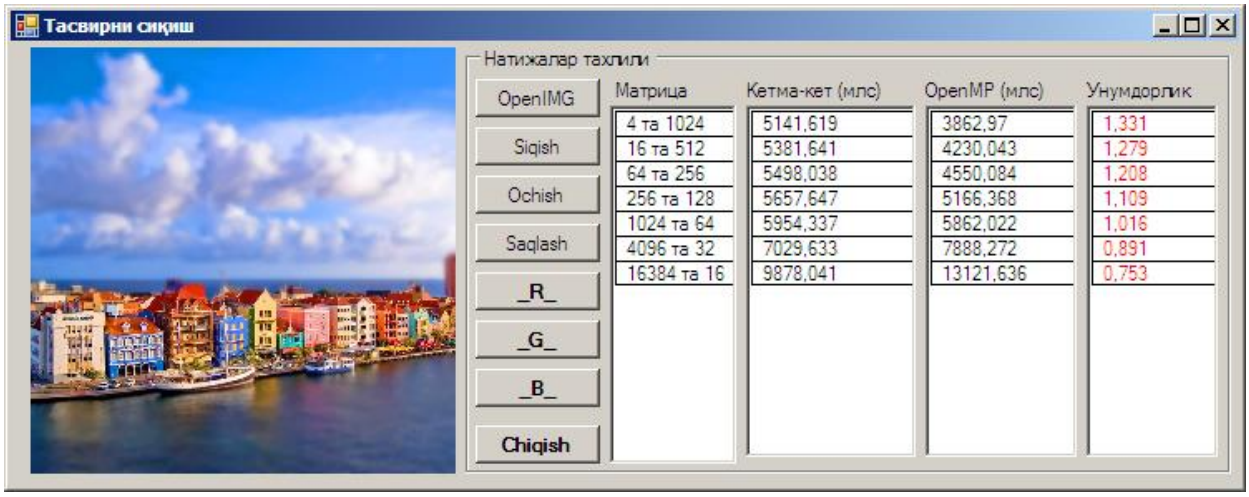


19-расм. Тасвирни сиқиш алгоритмининг амалий жараёни

Асосан сиқиш алгоритмлари сиқиладиган маълумотларда қандайдир қонунийлик мавжуд бўлганда яхши натижа беради (19-расм). Мисол учун тасвир маълумотлари ичида 0 лар сони 100 ни ташкил этади дийлик, бунда

тасвирни хотирада сақлашда фақатгина 100 сонини ёзиб қўйиш кифоя. Тасвирни хотирадан ўқиш давомида декодирлайдиган дастурлар 100 сонини ўқиган «0» лар кетма кетлиги деб тушунади. Айрим ҳолатларда 0 лар кетма кетлиги ўртасида 1 сони мавжуд бўлса унда сиқиш натижа бермаслиги мумкин. Чунки бу сонлар кетма кетлигини 2 та қиймат билан ифодалашга тўғри келади. Ифодалар сонининг ошиши ҳар бир ўзгаруичи учун қўшимча хотирадан жойни талаб қилади ва хотирадан ўқиш вақтида ўзгарган қиймат учун процессор навбатига яна қўшимча амаллар кетма кетлигини юклашга тўғри келади. Бу эса амал бажариш қонунига асосан қўшимча вақт талаб қилади.

Лекин яна бир савол туғилади: нега айнан тасвирнинг ҳар бир деталини сақлашимиз керак? Инсон тасвирга қараганда унда нима акс этганини англай олади, ундаги ёруғликнинг тебраниш даражасини илғай ҳам олмайди, шундан келиб чиқиб тасвирдаги айрим кўз илғамас қийматларни бошқа сон билан ифодаласа ҳам бўлади. Шунда тасвир маълумотларини сиқанимизда яхшироқ натижага эришамиз.



20 - расм. Тасвирни Вейвлет жараён орқали сиқишнинг дастурий кўриниши

Юқорида келтириб ўтилган назарий маълумотларга асосан тасвирларни Вейвлет жараён орқали сиқишда параллелаштириш

алгоритмларидан фойдаланилди. Алгоритмларнинг амалиёти сифатида Microsoft Visual Studio 2010 дастурий пакетида Visual C++ (v100) [23,26,30] и Intel C++ Compiler XE 11.0 компиляторидан фойдаланган ҳолда дастур тузилди. C++ кутубхоналаридан фойдаланилди, шунингдек кўп оқимлиликни ташкиллаштириш сифатида OpenMP компилятор директивасидан фойдаланилди.

Дастурда кетма кет ва оқимларга ажратиш натижалари ва улар орасида эришилган унумдорлик даражасини кўриш учун 4 та натижалар ойнаси келтирилган. Бундан ташқари дастурни ишга туширгач қайта ишланаши керак бўлган тасвирни юклаш учун тугма қўйилган. Қуйидаги расмда дастурнинг асосий ойнаси келтирилган.

Дастурни ишга тушуриш учун фойдаланувчи компьютерига қуйидаги минимал тизим талаблари қўйилади:

- процессор тактик частотаси 900 МГц ёки ундан ортиқ бўлиши лозим (дастурда параллелаштириш алгоритмларидан фойдаланиш ва ундан унумдорлик натижасига эришиш учун кўп ядроли процессор ишлатилади);
- 512 МБ ОХҚ;
- Айланиш частотаси 5400 айл/мин га тенг қаттиқ диск;
- Доимий хотирадан 200 МБ бўш жой;

Дастурий таъминот учун қуйидаги талаблар:

- операцион тизим
  - Windows 8 ёки Windows 7;;
  - Windows Vista қайта ишланган охирги 2 (SP2) версияси;
  - Windows Server 2008 қайта ишланган охирги 2 (SP2) версияси;
  - Windows Server 2003 қайта ишланган охирги 2 (SP2) версияси;
  - Windows Server 2003 R2 ёки қайта ишланган охирги версияси;
  - Windows XP с қайта ишланган 3 (SP3) версияси.

- Intel® C++ Compiler 9.0 (ёки янги) для Windows ёки Microsoft VisualC++ 7.1.

Дастурни ишга тушуриш учун Microsoft Visual C++ 2010 дастурий мажмуасининг бер неча пакетларини ўрнатиш зарур бўлади. Бу пакетлар қуйидаги библиотекаларни: C Runtime (CRT), Standard C++, ATL, MFC, OpenMP ва MSDIA ни ишлашини таъминлайди.

Дастурга тасвир юклангач, уни сиқин тугмасиги босиш орқали Вейвлет жарайнини амалга оширилиди ва тасвир маълумотларини сиқиш амалга оширилади. Амаллар кетма кетлиги қуйидагилардан иборит:

- Дастурга юкланган тасвирнинг RGB – қийматларини байтли массивга юклаб олинади;
- RGB – қийматларни квантланган умумий ранглар компаненталарини YCrCb га кодланади;
- Вейвлет жараёни амалга оширилади;
- Кўп ўлчамли массивлар қаторини бир ўлчамли массивга ўзлаштириб оламиз;
- Ихтиёрий сиқиш алгоритми орқали ҳосил бўлган маълумотлар қаторини сиқамиз.

Юқоридаги босқичларнинг биринчи босқичида биз тасвир ўлчамлари стандарт бўлмаган холат учун ташкил этилди. Тасвир дастурга юклангач унинг ўлчамлари Visual C++ дастурлаш тилида мавжуд функсиялар орқали ўзлаштириб олинади.

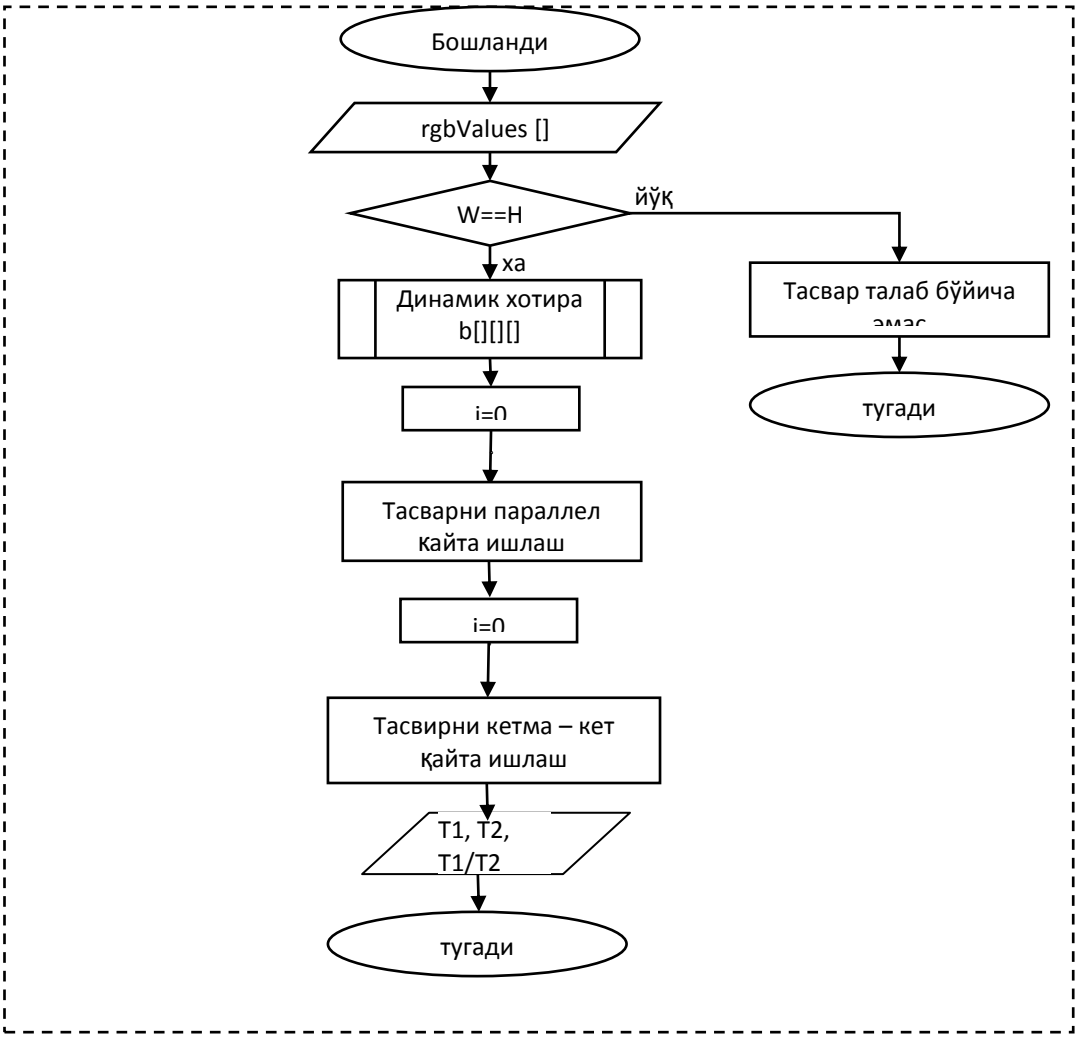
```
Bitmap^ bmp = gcnew Bitmap(pictureBox1->Image);
BitmapData^ bmpData = bmp->LockBits(Rectangle(Point(), bmp->Size),
ImageLockMode::ReadOnly, PixelFormat::Format24bppRgb);
int width = bmp->Width; (тасвир энининг ўлчами)
int height = bmp->Height; (тасвир бўйининг ўлчами)
```

Ўзлаштириб олинган қийматлардан келиб чиқан ҳолда динамик хотира ҳосил қилинади. Динам хотира ҳосил қилишдан ососий мақсад

шунданки, тасвир маълумотлари қайта ишлаш давомида ўз қийматларини ўзгартиради ва маълумотлар хажмига тенг хотирадан жой ажратилади:

```
unsigned char**** b = new unsigned char***[mat];
          for(int k = 0; k < mat; k++)
          {
               b[k] = new unsigned char**[3];
               for (int i = 0; i < 3; i++)
               {
                    b[k][i] = new unsigned char*[width];
                    for (int j = 0; j < width; j++)
                    {
                    b[k][i][j] = new unsigned char[height];
                         }
                    }
               }
```

Энди эса дастурнинг ишлаш тартиби билан танишиб ўтамиз. Дастурнинг алгоритм қисми қуйидаги кўринишга эга:



21 – расм. Тасвирни сиқиш дастурининг блок схемаси

Вейвлет жараённи амалга оширганимизда ананавий Jpeg алгоритмидан фойдаланилди. Бунда тасвир матрицасини бошланғич ўлчами сифатида 8x8 деб олинди.

2. Бир ва кўп ядроли процессорларда бажариладиган амалларни оқимларга ажратиш алгоритмлари

Тасвир маълумотларини вейвлет жараён орқали қайта ишлаганимизда бир – бирига боғлиқ бўлмаган жараёнлар кетма кетлигини процессорга юкланишини кўришимиз мумкин. Бунда айрам ҳолларда бир хил амаллар кетма кетлиги навбат билан қайта ишланилиши вақтдан йўқотилишга олиб келди. Ўртача ҳолатда 2048x2048 ўлчамга эга рангли тасвирни қайта ишлаганимизда 2 ядроли процессорда тахминан 1251,653 млс ва 4 ядроли процессорда эса 387,37 млс вақт кетишини кўришимиз мумкин. Бу балким унча катта бўлмаган қийматдир, аммо мултимедиа тизимларида маълумотлар кетма кетлигидан ташкил топган анимацияни қайта ишлаганда сезиларли даражада кутиб қолишларни гувоҳи бўлишимиз мумкин.

Ҳозирги вақтгача ишлаб чиқарилган дастурий воситалар асосан бир ядро процессорлар учун тадбиғ этилган эди. Ахборот технологияларнинг тўхтовсиз равишда ривожланиб бораётгани, анаънавий алгоритмларнинг унумдорлик даражасининг нисбатан пасайишини кўрсатмоқда. Бунга асосий сабаб ахборот хажмининг кескин ошиши ва маълумотларнинг тузилиш жихатидан мураккаблашиб бораётганлигидадир. Ҳозирга қадар яратилган дастурий воситалар фақатгина маълум белгиланган чукланишлар эвазига амаллар кетма кетлигини бажаришар эди, бу қандайдир шартларни қаноатлантира оларди. Айниқса мультимедиа тизимларида ахборотга ишлов бериш мураккаб жараёнлардан ташкил топади. Оддийгина тасвир устида бажариладиган қайта ўзгартириш амалларини оладиган бўлсак, авволо хотира масаласига дуч келамиз.

Тасвир сифатининг яхшиланиб бориши ва тасвир маъносининг мураккаблиги унинг хажмига ҳам таъсир этмасдан қолмайди. Бунда хотирадан тасвир маълумотларини ўқиш ва қайта ишланган натижаларни ёзиш каби амаллар иш кўламининг асосий вақтини сарфлаб қўяди. Албатта процессорда амалга ошадиган жараёнлар юқоридаги муаммолардан қолишмайди. Хотирадан процессорга юклаш жараёнини амалга оширишда шинанинг бўшаш вақтини пойлаш ва банд қилиш вақти тақсимлаш хам тизим томонидан чекланганлиги тасвир усида бажарилиши керак бўлган ишни секинлашишига олиб келадиган сабаблардан биридир. Айниқса процессор билан боғлиқ томонлари ҳам муҳим ҳам мураккаб жараён ҳисобланади.

Процессорларда кўп ядролик тушунчасининг пайдо бўлиши «кўп оқимлилик» тушунчасини ҳам дастурий таъминотда кўпроқ тилга олинишга сабаб бўлди. Бу икки тушунча ҳозирги ахборот технологияларда узвий боғлиқ «жуфтлик» ҳисобланади.

Юқорида келтирилган дастурда ҳам тасвирлар устида қайта ишлаш амалларини оқимларга ажратишнинг икки хил усули қўлланилган. Ҳар бир усулда ҳам OpenMP нинг оқимларга ажратиш директивалари қўлланилган. Бу усулларни новбат билан таҳлил қилиб чиқамиз.

Биринчи усулда тасвирдан олинган маълумотлар массивини вейвлет жараён орқали қайта ишлаганимизда бир – бирига боғлиқ бўлмаган амаллар кетма – кетликларни ажратиб оламиз ва шу амалларни оқимларга ажратиш директиваларга бирлаштирамиз. Изоҳ сифатида шуни такидлаш лозим: агар параллелаштирилаётган амаллар кетма – кетликлар умумий боғлиқ ўзгарувчига боғланган бўлса, оқимларга ажратиш жараёнида дастур хатоликлари пайдо бўлиши мумкин. Мисол сифатида дастурда қўлланилган тасвирнинг RGB – қийматлари байтли массивга ўзлштириш жараёнини кўриб чиқсак:

```
#pragma omp parallel for shared(rgbValues) // параллел жараённинг бошланиши
    for(int k = 0; k < mat; k++)
```

```
     {
      if( k % N == 0 ) { s1=0; s2++; }
      for (int i = 0; i < width; i++)
     {
          for (int j = 0; j < height; j++)
          {
b[k][0][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 0];
b[k][1][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 1];
b[k][2][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 2];
          }
     } s1++;
}
#pragma omp barrier // параллелаштириш жараёни синхрон ҳолда якунланади
}
```

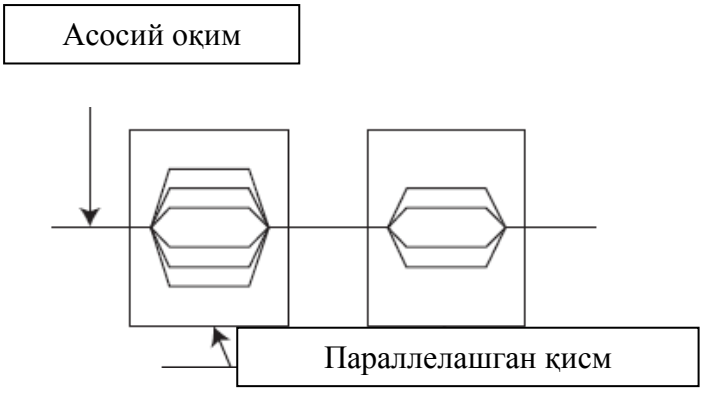
OpenMP да синхронизация. Келтирилган кодда ***s1***ва ***s2*** ўзгарувчилар **rgbValues** массив учун умумий ўзгарувси бўлиб қолган, бу эса коднинг бошланишида эълон қилинган #pragma omp parallel for директиванинг shared(rgbValues) шартини қаноатлантирмайди. Бу ҳолатни назарда тутган ҳолда шартни қаноатлантирувчи қўшимча ўзгарувчилар массивини қўшиш ва цикл ичидан умумий бўлган ўзгарувчиларни олиб ташлаш кифоя. Бунинг натижасида ўзгартириш киритилган кодлар кетма кетлиги қуйидаги кўринишда бўлади:

```
int s1[] = { 0,1,0,1 };
int s2[] = { 1,1,2,2 };

#pragma omp parallel for shared(rgbValues) // параллел жараённинг бошланиши
     for(int k = 0; k < mat; k++)
     {
      for (int i = 0; i < width; i++)
     {
          for (int j = 0; j < height; j++)
          {
b[k][0][i][j] = rgbValues[j + (s1[k] * width) * width * 3 + i + (s2[k] * width) * 3 + 0];
b[k][1][i][j] = rgbValues[j + (s1[k] * width) * width * 3 + i + (s2[k] * width) * 3 + 1];
b[k][2][i][j] = rgbValues[j + (s1[k] * width) * width * 3 + i + (s2[k] * width) * 3 + 2];
          }
```

```
    }
#pragma omp barrier // параллелаштириш жараёни синхрон ҳолда якунланади
```

Ихтиёрий кетма - кет ёки параллел дастур икки хил қисмлар тўпламидан иборат бўлади: кетма - кет қисм ва параллел қисм. Кетма - кет бажариладиган қисмда фақат битта бош оқим (мастер) яратилади. Ушбу оқимда дастур инициализация қилинади, шунингдек, дастурнинг якунланиши ҳам ушбу оқим орқали бўлади. Кетма -кет бажариладиган дастурда параллеллаштириш қисми учраганда фақат битта бош оқим яратилади ва ушбу оқим дастурнинг бажарилиши давомидаги ягона оқим бўлиб қолади. Параллел бажариладиган дастурда параллеллаштириш қисми учраганда бир қанча параллел оқимлар (slave) яратилади. Яратилган параллел оқимлар турли хил процессорли ёки битта процессорли ҳисоблаш тизимларида бажарилиши мумкин. Битта процессорли ҳисоблаш тизимларида параллел оқимлар процессордан жой олиш учун ўзлари ўртасида рақобатлашади.Рақобатлашишнибошқаришоперацион тизимнинг режалаштирувчиси томонидан махсус алгоритмлар ёрдамида амалга оширилади. Линух операцион тизимидаги вазифаларни режалаштирувчи процессларни қайта ишлашда стандарт ҳисобланган арғумчоқ алгоритми (роунд - робин)дан фойдаланади. Бунда фақат тизим администраторлари ушбу алгоритми тизим воситалари ёрдамида ўзгартириши мумкин. Параллел дастурнинг принсипиал схемаси 22- расмда келтирилган:



22- расм. Параллел дастурнинг схемаси

Параллел дастурнинг ишга туширилиши натижасида бош оқимнинг инициализация қисмидан ва унинг процесси бажарилишидан бошланади. Унда заруратга қараб параллел оқимлар яратилади ва уларга керакли маълумотлар узатилиб бажарилади. Параллел оқимлар дастурнинг битта параллел қисмида бир - биридан мустақил равишда ёки бир - бири билан алоқа ўрнатган ҳолда бажарилиши мумкин. Бир - бири билан алоқа ўрнатган ҳолда бажарилиши дастурнинг ишлаб чиқарилишини мураккаблаштириши мумкин.

Бундай ҳолларда дастурчи параллел оқимлар ўртасида маълумотлар узатилишини режалаштириши, ташкил этиши ва синхронизациялаши керак бўлади. Параллел дастурларни ишлаб чиқишда параллеллаштириш қисмларидаги параллел оқимларни мустақил равишда ишлашини таъминлаш мақсадга мувофиқ бўлади. Параллел оқимлар ўртасида маълумотлар алмашиши учун OpenMP да умумий ўзгарувчилар ишлатилади. Умумий ўзгарувчиларга турли параллел оқимлардан мурожаат қилинганда маълумотларга эга бўлишда конфликт ҳолати юзага келиши мумкин.Ушбу конфликтни олдини олиш учун синхронизация (synchronization) процедурасидан фойдаланиш мумкин. Синхронизация процедурасини бажаришга жуда кўп вақт кетишини ҳисобга олиш керак ва иложи бўлса ушбу процедурани кам ҳолларда ишлатиш мақсадга мувофиқ бўлар эди. Бунинг учун дастур тузишда маълумотлар структурасини жуда пухта ўйлашга тўгри келади.

OpenMP да маълумотлар модели. OpenMP да маълумотлар модели ҳамма оқимлар хотира муҳити учун умумий ва ҳар бир оқим учун локал хотира қисми мавжуд деб тахмин қилинади [8,11,12,15]. OpenMP да параллел муҳитдаги ўзгарувчилар 2 турга бўлинади:

- shared (умумий, ҳамма оқимлар ушбу турдаги ўзгарувчиларни кўради);
- private (локал, ҳар бир оқим ўзгарувчининг нусхасини ўзида кўради).

Умумий ўзгарувчи ҳамма қисмлар учун ҳар доим фақат битта нусхада бўлади ва барча оқимларга битта номда бўлади.Локал ўзгарувчилар эълон қилинганда, ҳар бир оқим учун бир хил типдаги ва ўлчамдаги нусхалари яратилади. Битта оқимдаги локал ўзгарувчининг қиймати ўзгарса ҳам қолган оқимлардаги нусхаларининки ўзгармайди.

Бундан ташқари дастурнинг код қисмида ҳар бир квадрат матрица устида параллелаштириш амаллари қўлланилган. For циклида оқимларга ажратиш #pragma omp parallel for дириктиваси билан амалга оширилади. Қўйидаги дастурий кодда унинг қўлланилиши келтирилган:

```
#pragma omp parallel for
for (j = 0; j < n; j++)
{
	xWavelet[j] = ImgArray[Component][dwPos][j];
}
```

Parallel директиваси ёрдамида параллел муҳит ҳосил қилинади. C/C++ дастурлаш тилида [22,25] қуйидагича кўринишда бўлади:

#pragma omp parallel [шарт [[,] шарт]…]

Юқорида келтирилганларга асосан қуйидагича хулоса қилиш мумкин: дастурни параллел қисмларга ажратилиши ва параллел процессларни ишлаб чиқиш муҳимдир.

3. Кўп ядроли процессорларда эришилган тескорликни таҳлил қилиш

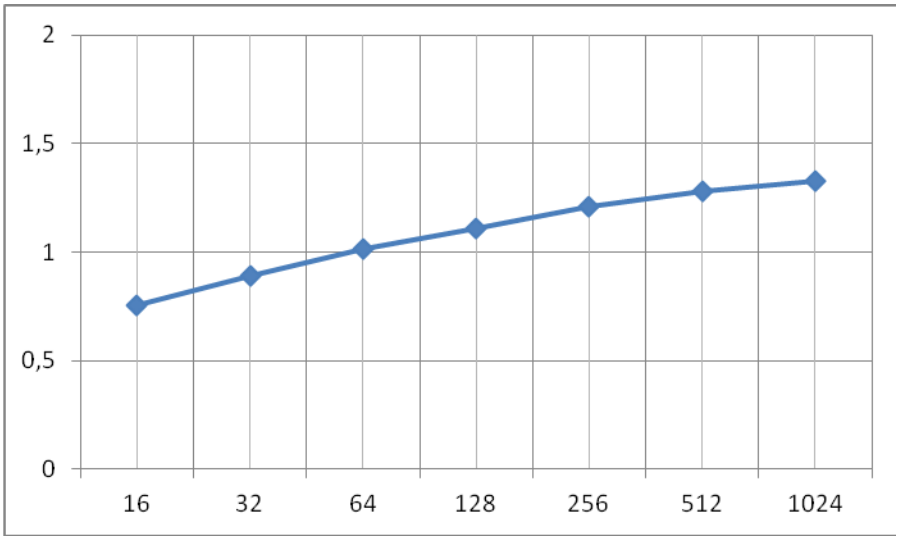
Мультимедиа тизимларида тасвирларни қайта иўлаганда вейлет жараёнларини тадбиғ этиш яхши натижа беради. Айниқса параллелаштириш алгоритмларидан фойдаланиш унумдорлик даражасини оширишга ёрдам беради. Тасвирларни қайта ишлаганда биринчи усул ёрдамида амалга оширамиз. Бу усулнинг амалга ошиш алгоритми қуйидагича амалга ошади: дастурга юкланган тасвир 2^N қиймат билан

амалга ошади. N нинг қиймати тасвир бўлинган матрицаси 16x16 ўлчамга эга бўлгунча амалга ошади. Бу ҳолатда мисол сифатида N = 1 га тенг бўлганда тасвир 4 та матрицага ажралада (3.2-расм) . Бундан кўринади алгоритм OpenMP дан фойдаланганда 2 ядроли процессорда амалга оширилгада 2 та оқимга 2 марта бўлиб берилади ва цикл 2 марта айланишга тўғри келади. Кетма кет амалга оширганда эса бу амаллар бажарилганда цикл 4 маротаба айланишга тўғри келади. Бу ҳолатни назарий жихатдан тахлил қиладиган бўлсак, унумдорлик 2 маротаба ошади деган хулосага келишимиз мумкин. Аммо OpenMP ёрдамида оқимларга ажратганда хотира ва процессор билан оқимларни ташкиллаштирганда маълум вақт сарфланади. Чунки оқим яратилганда хотирага ҳосил бўлаётган оқим учун динамик хотира яратиш лозим ва оқим ўз жараёнини якунлаганда динамик хотирани ўчириш керак бўлади. Кейинги жараён бу оқимларни процессорда бажарилиш учун навбатга қўйиш. Бу ҳолда ҳам маълум даражада вақт сарфланади. OpenMP пакетидан фойдаланган ҳолда тасвирни қайта ишлаганда 2 ядроли процессорда қайта ишлаганда қуйидаги 1 – жадвалда кўрсатилган натижаларга эришилди. Жадвалнинг биринчи устунида қайта ишланаётган тасвирнинг нечта матрицага бўлиниши ва бўлинган ҳар бир матрицанинг қандай ўлчамга эга эканлиги кўрсатилган.

1 – жадвал. 2 ядроли процуссорда тасвирларни қайта ишлаганда сарфланган вақт ва унумдорлик

N x N	OpenMP сиз	OpenMP ёрдамида	Унумдорлик
4 та 1024	5141,619	3862,97	1,331
16 та 512	5381,641	4230,043	1,279
64 та 256	5498,038	4550,084	1,208
256 та 128	5657,647	5166,368	1,109
1024 та 64	5954,337	5862,022	1,016
4096 та 32	7029,633	7888,272	0,891
16384 та 16	9878,041	13121,636	0,753

Натижалардан шуни кўришимиз мумкин, бир оқимли кетма кет қайта ишлаш амалга оширганда OpenMP ёрдамида амалга оширганга нисбатан кўпроқ вақт талаб қилиши мумкин.



23 – расм. OpenMP ёрдамида 2 ядроли процессорда эришилган унумдорликни график кўриниши

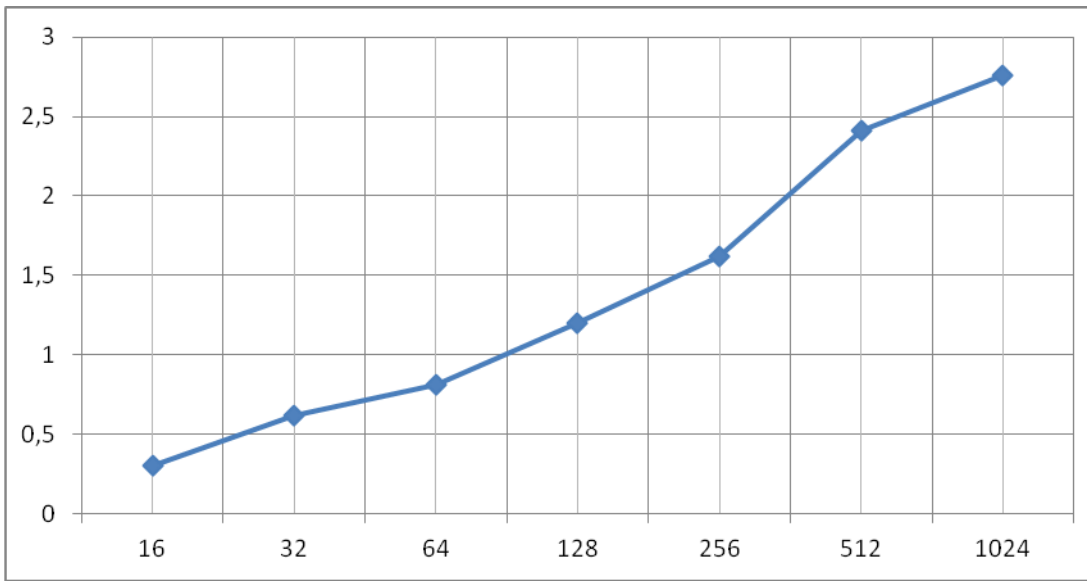
OpenMP ёрдамида тасвирни қайта ишлаганда *#pragma omp parallel* дириктиваси процессорни максимал ҳолда оқимларга ажратишга ҳаракат қилади. Яъни 2 ядроли процессорда максимал ҳолда 2 та оқимни яратиб бериши мумкин. Аммо оқимларни максимал ҳолда белгилаб бериш билан унумдорликка эришиб бўлмайди. Чунки оқимларни ташкиллаштиришга ҳам боғлиқ ҳисобланади. Бу вазифини OpenMP компилятори ташкиллаштириб беради. Унумдорлик натижасини 23 – расмда кўришимиз мумкин.

Айнан шу ҳолатни 4 ядроли процессорда амалга оширганимизда 2 – жадвалдаги натижаларни кўришимиз мумкин:

Ядролар сонининг ошиши оқимлар сонини ошишига имкон беради. 4 ядроли процессорда максимал ҳолда 4 та оқим ташкиллаштиришимиз мумкин (24-расм).

2 – жадвал. 4 ядроли процессорда тасвирларни қайта ишлаганда сарфланган вақт ва унумдорлик

N x N	OpenMP сиз	OpenMP	Унумдорлик
4 та 1024	1873,058	679,337	2,757
16 та 512	1693,052	701,696	2,413
64 та 256	1581,531	977,97	1,617
256 та 128	2017,086	1676,533	1,203
1024 та 64	1446,008	1776,122	0,814
4096 та 32	1630,208	2636,226	0,618
16384 та 16	2143,645	7029,12	0,305

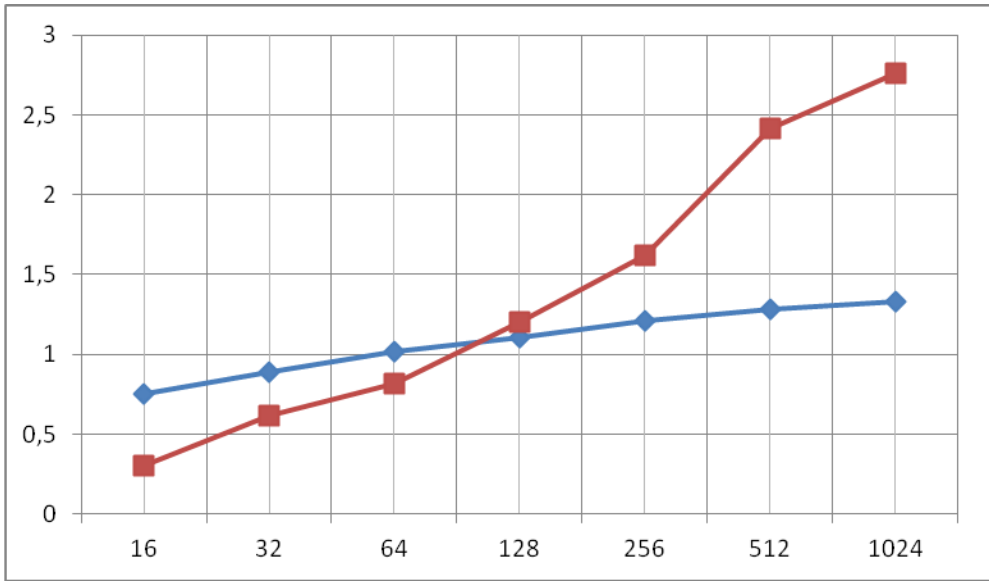


24– расм. OpenMP ёрдамида 4 ядроли процессорда эришилган унумдорликни график кўриниши

Унумдорлик натижасини 23– расмдаги графикда шуни кўришимиз мумкинки, тасвир бўлинган кесимлар ўлсами 64 дан кичкина бўлса алгоритм унумдорлик бермайди. Унумдорликни оптимал қийматлар деб 512-1024 оралиғи аниқланди.

Юқоридаги натижалардан шуни кўришимиз мумкинки, унумдорлик натижаси *n* ядроли процессорларда *n* дан ошмаслигини кўришимиз мумкин. 2 ва 4 ядроли процессорларни унумдорлигини солиштирганимизда қуйидаги 24-расмда фарқини кўришимиз мумкин.

Тасвирларни қайта ишлашда унумдорликни оширишнинг кейинги усулимизда алгоритм қуйидагича амалга ошади: тасвирни қайта ишлаганда авволи тасвир пикселини минимал ҳолда 64 та қийматини қайта ишлашни бошлайди ва максимал ҳолда тасвирниниг ўлчамигача оборади.



25– расм. OpenMP ёрдамида 4ва 2 ядроли процессорда эришилган унумдорликни график кўриниши(қизил-4 ядро, кўк-2 ядро)

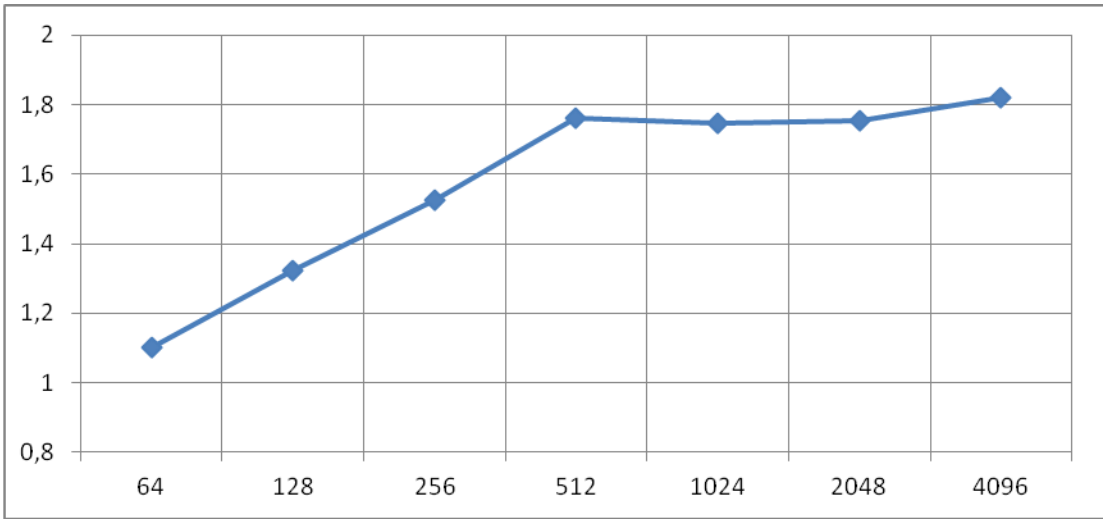
Тасвирни кесмаларга бўлмасдан зарраларни қайта ишланганда уларнинг бошланғич ўлчами сифатида 64 деб олинда ва 2^n бўйича давом эттирилди (3- жадвал).

3 – жадвал. 2 ядроли процессорда тасвирларни **nxn** ўлчамда қайта ишлагандаги унумдорлик

n x n	**Унумдорлик**
64	1,101
128	1,324
256	1,524
512	1,761
1024	1,747
2048	1,755
4096	1,819

Бу усулни 2 ядроли процессорда қўллаганимизда 3 - жадвалдаги натижаларга эришилди.

nxn қийматининг ўзгаришини қуйида келтирилган графикда яқол кўришимиз мумкин.



26 – расм. тасвирни **nxn** ўлчамдаги заррачаларга бўлиб қайта ишлаганда 2 ядроли процессорда эришилган унумдорлик

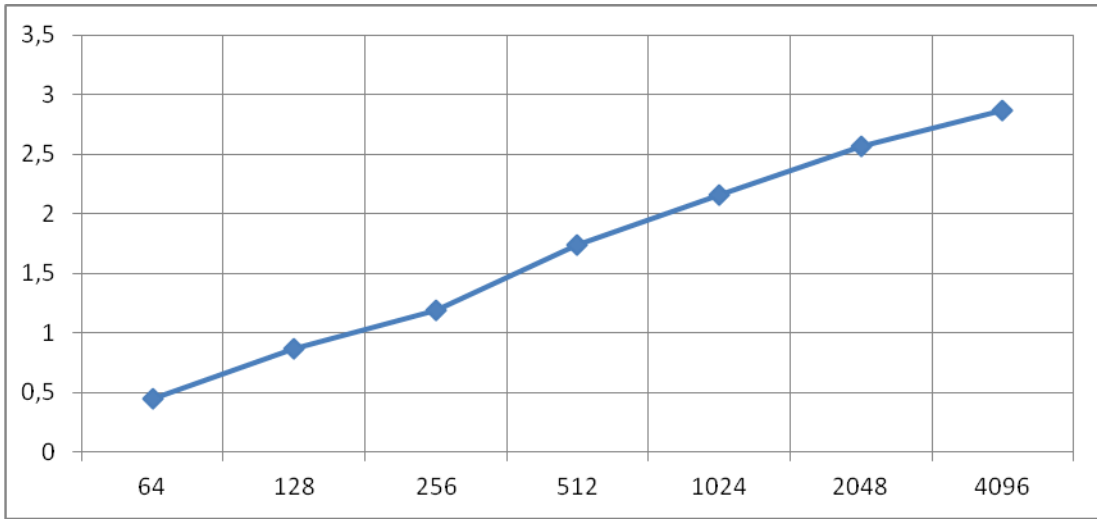
Графикдан шунидай хулоса қилиш мумкинки, тасвирнинг ***nxn*** зарралар сонини оширганимиз сари унумдорлик даражаси ошиб борар экан. Оптимал ечим сифатида 512 қиймати танланди, чунки тажриба натижалардан 512 қийматида процессор унумдорлиги яқол намоён бўлди.

4 – жадвал. 4 ядроли процуссорда тасвирларни **nxn** ўлчамда қайта ишлаганда эришилган унумдорлик натижалар

n x n	**Унумдорлик**
64	0,452
128	0,864
256	1,189
512	1,743
1024	2,157
2048	2,569
4096	2,869

Энди эса айнан шу усулни 4 ядроли процессорда амалга ошириб кўрдик. Унинг унумдорлик натижаси 4 жадвалда берилган.

Бунда ҳам натижани график кўринишида намойиш қилганимизда юқоридаги кўринишга яқин график пайдо бўлди:



27 – расм. тасвирни **nxn** ўлчамдаги зарраларга бўлган ҳолда қайта ишлаганда 4 ядроли процессорда эришилган унумдорлик

27 - расмдаги график кўринишидан 4 ядроли процессорларда тасвирни зарраларга бўлиб қайта ишлаганимизда унумдорлик даражаси кескин ўсиб бориш ҳолитини кўришимиз мумкин. Аммо графикда максимал ҳолда тасвир зарра ўлчами 4096 га тенглигини кўришимиз мумкин. Буниниг сабаби процессорнинг кеш хотирасининг хажмига боғликдир. Чунки тасвир зарраларини процессорда қайта ишлаганимизда кеш хотирадан вақтинчалик динамик хотира яратиш лозим бўлади. Бу эса тасвир маълумотларининг хажм жихатдан катталиги хотира етишмовчиликларини келтириб чиқаради.

III боб бўйича хулоса

Параллелаштириш алгоритмлар ёрдамида олинган тажриба таҳлил маълумотлари яхши натижалар кўрсаттики, таққослашга ва оптимал алгоритмни танлашга имкон берди. Бундан ташқари тажриба натижаларидан олинган маълумотларга асосланиб тасвирнинг қандай қиймалари орасида биз танлаган параллелаштириш усулининг яхши натижа бериши аниқланди. Юқоридаги натижалар асосида қуйидаги хулосалар кетма-кетлик рўйхатини келтиришимиз мумкин:

1. Тасвир маълумотларини параллелаштириш алгоритими билан қайта ишлаганимизда кўп ядроли процессор унумдорлигини қуйидаги хусусият – тасвирнинг қайта ишланилаётган қийматлар сонига боғлиқ экан, қайсики қийинчилик пойдо қиладиган юқорида келтирилган алгоритмларда. Процессор унумдорлиги етарли даражага етгунича ўсиб боради (тасвирнинг 256 дан 4096 гача). Оптимал ечим сифатида унумдорлик даражаси процессорларнинг ядролар сони қийматига яқинлашиб боради.

2. Параллел қайта ишлашда оқимлар сонининг оптимал сони процессорнинг хисоблаш даражасига тенг бўлиши лозим. Икта физик ядрога эга процессорлар 4 та мантиқий оқим яратиб бера олиши ва унумдорлик даражасини ҳақиқий 4 ядроли процессор натижасига яқинлаштириб бериши мумкин. Булар Intel Core i5 ва Core i3 процессорларни солиштириш ёрдамида кўриб чиқилди.

3. Яна бир муҳим хусусиятлардан бири шундаки, бир вақтнинг ўзида бажарилаётган операциялар сони ажратилган оқимлар сонига тенг бўлиши лозим.

4. Кўп ядроли процессорларда тасвирни кесмаларга ажратиб қайта ишланганда 512-1024 оралиқ оптимал қийматлар ҳисобланади. Тасвирни заррачалар сонини ошириш ёрдамида қайта ишлаганда 2 ядроли процессорда 512 ва 4 ядроли процессорла 4096 қиймат оптималдир.

Хулоса

Қуйидаги илмий ишда мультимедиа тизимларида ахборотни жойлаштириш, сақлаш ва саралаш амалларини оптималлаштириш, хотира ресурларидан фойдаланишда сиқиш алгоритмларининг оптимал усулларини яратиш ва тасвирларни қайта ишлашда кўп ядроли процессорларнинг унумдорлик даражасини оширишга мўлжалланган параллелаштириш алгоритмларини яратиш. Ишни бажариш давомида қуйидаги натижаларга эришилди:

1. Мультимедиа тизимларида ахборотни хотирада сақлаш муаммоли масалаларини таҳлил қилган ҳолда сиқиш алгоритмларининг оптимал усулларидан фойдаланиш яхши натижа бериши аниқланди. Бунда тасвирларни рақамли қайта ишлаганда вейвлет-жараёнидан фойдаланиш қайта ишлаш амалларида оптимал ёндошиш мумкинлиги аниқланди. Бунинг натижасида хотира ресурсларидан фойдаланишда камроқ жой эгаллашини ва ахборотни хотирадан ўқиш ва қайта ишлаб қайта хотирага ёзиш жараёнлари тезлаштиришга олиб келди.

2. Тасвирларни қайта ишлашда кўп ядроли процессорларга мўлжалланган параллелаштириш алгоритмларини қўллаш яхши самара берди. C++дастурлаш тилидан фойдаланилди ва параллелаштиришни ташкиллаштириб берувчи OpenMP компилятори директиваларидан фойдаланилди ва улар ёрдамида процессор унумдорлик даражаси ошди.

3. Иш давомида тасвирларни қайта ишлашда тасвир қийматларини байтли массивга ўзлаштириш, вейвлет-жараёнларни амалга оширганда оқимларга ажратиш усуллари ва хотирани параллел ҳолда динамик жой ажратиш каби жараёнлар бажарилди ва яхши самарадорлик кўрсатди.

4. Тадқиқот натижаларидан келиб чиққан маълумотлар ёрдамида параллелаштиришнинг умумий усули яратилди. Тасвирлар заррачалар

сонини ошириш натижасида параллелаштириш алгоритмлари эффективлигини кўрсатти, баъзида эса бутун тасвирни квадрат қисмларга бўлган ҳолда алоҳида функцияларга ажратиш ва процессор оқимлар сонига тенг амалларга бўлиб бериш ҳам ўз унумдорлигини кўрсатти.

5. Тасвир маълумотларини параллелаштириш алгоритими ёрдамида оқимларга ажратиш билан қайта ишлаганимизда кўп ядроли процессор унумдорлигини қуйидаги хусусият – тасвирнинг қайта ишланилаётган қийматлар сонига боғлиқ эканлиги натижалардан келиб чиққан ҳолда аниқланди. Процессор унумдорлиги етарли даражага етгунича ўсиб боради (тасвирнинг 256 дан 4096 гача). Оптимал ечим сифатида унумдорлик даражаси процессорларнинг ядролар сони қийматига яқинлашиб боради.

6. Параллел қайта ишлашда оқимлар сонининг оптимал сони процессорнинг хисоблаш даражасига тенг бўлиши лозим. Икта физик ядрога эга процессорлар 4 та мантиқий оқим яратиб бера олиши ва унумдорлик даражасини ҳақиқий 4 ядроли процессор натижасига яқинлаштириб бериши мумкин.

Қуйидаги илмий ишда ядролар сони 2 ва 4 га тенг бўлган турли процессорларда ўтқазилган тажриба натижалари келтирилди. Натижаларга таянган ҳолда хулосалар чиқарилди.

Ишнинг натижаси – кетма-кет ва параллел алгоритмларни тасвирларни вейвлет-жараён ёрдамида сиқишда кетган вақтини ва унумдорлик даражасини аниқлайдиган дастурий восита яратилди. Дастур 2 ва 4 ядроли Intel Corei3 и i5 процессорларида Windows XP и Windows 7 операцион тизимларда тажрибалар ўтказилди. Қуйидаги дастур ёрдамида кўпгина тажрибалар ўтқазилди ва тажриба натижалар асосида якуний хулосалар чиқарилди.

Фойдаланилган адабиётлар

РЎЙХАТИ

Норматив-ҳуқуқий ҳужжатлар:

1. Ўзбекистон Республикасининг «Ахборотлаштириш тўғрисида»ги №560-II сонли қонуни. 11 декабря 2003 йил.
2. Ўзбекистон Республикаси Президентининг “Замонавий ахборот-коммуникация технологияларини янада кенгроқ жорий қилиш ва ривожлантириш чора-тадбирлари тўғрисида”ги қарори. 21 март 2012 йил.
3. И.А. Каримов. Мамлакатимизни модернизация қилиш йўлини изчил давом эттириш – тараққиётимизнинг муҳим омилидир/ Президент Ислом Каримовнинг Ўзбекистон Республикаси Конституцияси қабул қилинганининг 18 йиллигига бағишланган тантанали маросимдаги маърузаси. Ташкент, 07.12.2010.

Дарслик ва ўқув қўлланмалар:

4. Королев Л.Н. Архитектура ЭВМ. – М.: «Научный мир», 2005. 272 б.
5. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб: «БХВ-Петербург», 2002. – 608 б.
6. Богачев К.Ю. Основы параллельного программирования. – М.: «БИНОМ. Лаборатория знаний», 2003. – 342 б.
7. Малышкин В.Э.. Основы параллельных вычислений: Учебное пособие. Часть 2. – Новосибирск: ЦИТ СГГА, 2003. – 264 с.
8. Шпаковский Г.И. Реализация параллельных вычислений: MPI, OpenMP, кластеры, грид, многоядерные процессоры, графические процессоры, квантовые компьютеры. – Минск: Белорусский Государственный Университет, 2010. – 155с.

9. Ярославский Л.П. «Введение в цифровую обработку изображений», Москва сов.радио, 1979й.
10. Грузман И.С. «Цифровая обработка изображений в информационных системах», Новосибирск 2002г.
11. Антонов А.С. Параллельное программирование с использованием технологии OpenMP. – М: издательство Московского Университета, 2009 г. – 77с.
12. Левин М.П. Параллельное программирование с использованием OpenMP: учебное пособие. – М: “Бином. Лаборатория знаний”, 2008г. – 118 с.
13. Старченко А.В., Есаулов А.О. Параллельные вычисления на многопроцессорных вычислительных системах. – Томск: Изд-во ТГУ, 2009. – 56 б.
14. Мелехин В. Павловский Е. Вычислительные машины, системы и сети. – Масква, Изд-во: «Академия», 2007. – 103б.
15. Шпаковский Г.И. Реализация параллельных вычислений: MPI, OpenMP, кластеры, грид, многоядерные процессоры, графические процессоры, квантовые компьютеры. – Минск: Белорусский Государственный Университет, 2010. – 155б.

Илмий журналлардаги мақолалар:

16. Кардашев М.С. Масштабируемость параллельных алгоритмов цифровой обработки сигналов» / Сборник докладов республиканской научно-технической конференции молодых ученых, исследователей, магистрантов и студентов «Информационные технологии и проблемы телекоммуникаций», – Ташкент, 2013. – 65–67 с.
17. М.Ф. Рахимов. «Мультимедиа тизимларида тасвирларни параллел қайта ишлаш алгоритми»./ “Ахборот технологиялари ва

телекамуникация тизимларини самарали ривожлантириш истиқболлари” номли Республика илмий-техник конференцияси. – Тошкент, 2014. - I қисм 198-199 б.

18. Рахимов М.Ф. «Ўқув жараёни сифатини оширишда Flash технологиясидан фойдаланган ҳолда мультимедияли дарсликлар яратиш»./ “Алоқа ва ахборотлаштириш соҳаси учун кадрлар таёрлаш сифатини ошириш муаммолари” номли Тошкент ахборот технологиялари университети ва филиаллари профессор ўқитувчиларининг илмий услубий конференцияси. – Тошкент, 2014. – I том 99-100-101 б.

Интернет сайтлари:

19. Левин М.П.: «Параллельное программирование с OpenMP» / Левин М.П., режим доступа: http://www.intuit.ru/department/se/openmp/
20. http://www.intuit.ru/department/supercomputing/ppmcp/class/free/status/ (Параллельное программирование для многоядерных процессоров – курс INTUIT).
21. Intel Parallel Studio – инструмент для создания параллельных приложений. – http://www.ixbt.com/soft/intel-parallel-studio.shtml
22. C++ – OpenMP – Examples of basic parallel programming. – http://berenger.eu/blog/2010/07/23/c-openmp-examples-of-basic-parallel-programming.html
23. http://www.intuit.ru/department/calculate/inparallprog/class/free/status/ (Введение в методы параллельного программирования – курс INTUIT).
24. http://www.intuit.ru/department/supercomputing/ppinteltt/class/free/status/ (Параллельное программирование с использованием инструментов и технологий Intel – курс INTUIT).
25. http://www.microsoft.com/ru-ru/download/details.aspx?id=5555 (Распространяемый пакет Microsoft Visual C++ 2010).

26. http://software.intel.com/ru-ru/articles/intel-parallel-studio-parallelism-toolkit (Intel® Parallel Studio – инструмент для создания параллельных приложений).
27. http://habrahabr.ru/company/intel/blog/82486/ (Параллельные заметки – технология OpenMP. Блог компании Intel на Habrahabr).
28. http://www.openmp.org/drupal/node/view/8 (OpenMP: A Proposed Industry Standard API for Shared Memory Programming).
29. http://www.openmp.org/drupal/node/view/8 (OpenMP: C and C++ Application Program Interface Version 2.5).
30. http://habrahabr.ru/blogs/system_programming/121925/ (Основы MPI).

Илова

```
#pragma once
#include "Algorithms.h"
#include <omp.h>

namespace ImageGZ {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Diagnostics;
	using namespace System::Drawing;
	using namespace System::Drawing::Imaging;
	using namespace System::IO;
	using namespace System::IO::Compression;
	using namespace System::Runtime::InteropServices;
	using namespace System::Windows::Forms;

	/// <summary>
	/// Сводка для Form1
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		int b1, e1;
	public:
		int WV_TOP_TO_BOTTOM;
```

```
        Form1(void)
        {
            InitializeComponent();
            WV_LEFT_TO_RIGHT = 0;
            WV_TOP_TO_BOTTOM = 1;
        }
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^  e)
         {
if(openFileDialog1->ShowDialog()==
System::Windows::Forms::DialogResult::OK)
            {
            System::IO::StreamReader ^ sr = gcnew
        System::IO::StreamReader(openFileDialog1->FileName);
    Bitmap^ bmp = gcnew Bitmap(openFileDialog1->FileName);
                    pictureBox1->Image = bmp;
                        b1 = bmp->Height;
                        e1 = bmp->Width;
                }
         }

#include "Algorithms.h"

private: System::Void button2_Click(System::Object^ sender,
System::EventArgs^  e)
         {
            richTextBox1->Text = " ";
            richTextBox2->Text = " ";
            richTextBox3->Text = " ";
            richTextBox4->Text = " ";
```

```
Bitmap^ bmp = gcnew Bitmap(pictureBox1->Image);
BitmapData^ bmpData = bmp->LockBits(Rectangle(Point(), bmp->Size), ImageLockMode::ReadOnly, PixelFormat::Format24bppRgb);
IntPtr ptr = bmpData->Scan0;
Algorithms^ s1 = gcnew Algorithms();
int bytes = bmpData->Stride * bmp->Height;
array<Byte>^rgbValues = gcnew array<Byte>(bytes);
System::Runtime::InteropServices::Marshal::Copy( ptr, rgbValues, 0, bytes );
bmp->UnlockBits( bmpData );
int itr = bmp->Height/8;
int count = 0;
for(;;)
{
    if(itr / 2 == 1) break;
    count ++;
    itr = itr / 2;
}
int S;
for(S = 0; S <= count; S++)
{
int N = Math::Pow(2, S);
int width = bmp->Width/N;
int height = bmp->Height/N;
int mat = N * N;
richTextBox4->Text += mat + " ta " + Convert::ToString (width) + "\n";
unsigned char**** b = new unsigned char***[mat];

for(int k = 0; k < mat; k++)
{
```

```
                b[k] = new unsigned char**[3];
                for (int i = 0; i < 3; i++)
                {
                        b[k][i] = new unsigned char*[width];
                        for (int j = 0; j < width; j++)
                        {
                                b[k][i][j] = new unsigned char[height];
                        }
                }
        }
        double experimentsCount = 1;

        for (int exp = 0; exp < experimentsCount; exp++)
        {
                int s1 = 0, s2 = 0;
#pragma omp parallel for
                        for(int k = 0; k < mat; k++)
                        {
                                if( k % N == 0 ) { s1=0; s2++; }
                        for (int i = 0; i < width; i++)
                        {
                                for (int j = 0; j < height; j++)
                                {
b[k][0][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 0];
b[k][1][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 1];
b[k][2][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 2];
                                }
                        } s1++;
                }
#pragma omp barrier             }
```

```
double t0 = omp_get_wtime();
unsigned char **o = new unsigned char*[mat];
#pragma omp parallel for
for(int k = 0; k < mat; k++)
{
    o[k] = s1->Compress(b[k], width, height);
}
double t1 = omp_get_wtime();
double time_mseconds_par = double(t1-t0) * 1000 / experimentsCount;
richTextBox3->Text += String::Format("{0}", Math::Round(time_mseconds_par, 3)) + "\n";
    for (int exp = 0; exp < experimentsCount; exp++)
    {
        int s1 = 0, s2 = 0;
        for(int k = 0; k < mat; k++)
        {
            if( k % N == 0 ) { s1=0; s2++; }
        for (int i = 0; i < width; i++)
        {
            for (int j = 0; j < height; j++)
            {
b[k][0][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 0];
b[k][1][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 1];
 [k][2][i][j] = rgbValues[j + (s1 * width) * width * 3 + i + (s2 * width) * 3 + 2];
            }
        } s1++;
    }
    }
    t0 = omp_get_wtime();
```

```
                unsigned char **q = new unsigned char*[mat];
                for(int k = 0; k < mat; k++)
                {
                q[k] = s1->Compress1(b[k], width, height);
                }
                t1 = omp_get_wtime();

double time_mseconds_serial = double(t1-t0) * 1000 / experimentsCount;
richTextBox1->Text += String::Format("{0}", Math::Round(time_mseconds_serial, 3)) + "\n";
double acceleration = time_mseconds_serial / time_mseconds_par;
richTextBox2->Text += String::Format("{0}", Math::Round(acceleration, 3)) + "\n";
            int i, j, k;
            for( k = 0; k < mat; ++k)
                {
                for(i = 0; i < 3; ++i)
                    {
                        for( j = 0; j < width; ++j)
                        {
                            delete[] b[k][i][j];
                            b[k][i][j] = 0;
                        }
                        delete[] b[k][i];
                        b[k][i] = 0;
                    }
                    delete[] b[k];
                    b[k] = 0;
                } delete[] b;
            b = 0;
```

```
                    }
                }
private:    System::Void    button8_Click(System::Object^    sender,
System::EventArgs^  e)
 {
Application::Restart();
}
};
}

#include "StdAfx.h"
#include "Algorithms.h"
#include <math.h>
#include <omp.h>

using namespace System;
using namespace System::Drawing;
using namespace System::Drawing::Imaging;
using namespace System::Runtime::InteropServices;
Algorithms::Algorithms(void)
{
}
double round(double number)
{
   return number < 0.0 ? ceil(number - 0.5) : floor(number + 0.5);
}
unsigned char* Algorithms::Compress(unsigned char*** rgb, int cW, int cH)
{
        int f = 2;
        int dwDiv[8] = {48, 32, 16, 16, 24, 24, 1, 1};
```

```
int dwTop[8] = {24, 32, 24, 24, 24, 24, 32, 32};
int SamplerDiv = f, SamplerTop = f;
int YPerec = 100, crPerec = 85, cbPerec = 85;
int WVCount = 7;
double ***YCrCb = YCrCbEncode(rgb, cW, cH, YPerec, crPerec, cbPerec, cW, cH);
for (int z = 0; z < 3; z++)
{
    for (int dWave = 0; dWave < WVCount; dWave++)
    {
        int waveW = static_cast<int>(cW / pow(2, (double)dWave));
        int waveH = static_cast<int>(cH / pow(2, (double)dWave));
        if (z == 2)
        {
            YCrCb = WaveletePack(YCrCb, z, waveW, waveH, dwDiv[dWave], dwTop[dWave], dWave);
        }
        else
        {
            YCrCb = WaveletePack(YCrCb, z, waveW, waveH, dwDiv[dWave] * SamplerDiv, dwTop[dWave] * SamplerTop, dWave);
        }
    }
}
unsigned char *flattened = DoPack(YCrCb, cW, cH, WVCount);
for(int i = 0; i < 3; ++i)
    {
        for(int j = 0; j < cW; ++j)
        {
            delete[] YCrCb[i][j];
```

```
                YCrCb[i][j] = 0;
            }
            delete[] YCrCb[i];
            YCrCb[i] = 0;
        }delete[] YCrCb;
    YCrCb = 0;
    return flattened;
}

unsigned char *Algorithms::DoPack(double*** ImgData, int cW, int cH, int wDepth)
    {
        short Value;
        int lPos = 0;
        int size = cW * cH * 3;
        int intCount = 0;
        array<short>^shorts = gcnew array<short>(size);
        array<unsigned char>^Ret = gcnew array<unsigned char>(size);
        for (int d = wDepth - 1; d >= 0; d--)
        {
            int wSize = static_cast<int>(pow(2.0, (double)d));
            int W = cW / wSize;
            int H = cH / wSize;
            int w2 = W / 2;
            int h2 = H / 2;
            if (d == wDepth - 1)
            {
#pragma omp parallel for private(Value)
                for (int z = 0; z < 3; z++)
                {
```

```
                    for (int j = 0; j < h2; j++)
                    {
                        for (int i = 0; i < w2; i++)
                        {
Value = static_cast<short>(round(ImgData[z][i][j]));
 ((Value >= -127) && (Value <= 127))
                                {
Ret[lPos++] = static_cast<unsigned char>(Value + 127);
                                }
                                else
                                {
Ret[lPos++] = (unsigned char)255;
shorts[intCount++] = Value;
                                }
                        }
                    }
                }
            }
#pragma omp parallel for private(Value)
            for (int z = 0; z < 3; z++)
            {
                for (int j = 0; j < H; j++)
                {
                    for (int i = w2; i < W; i++)
                    {
Value = static_cast<short>(round(ImgData[z][i][j]));
                        if ((Value >= -127) && (Value <= 127))
                        {
                Ret[lPos++] = static_cast<unsigned char>(Value + 127);
                        }
```

```
                                    else
                                    {
                    Ret[lPos++] = static_cast<unsigned char>(255);
                                            shorts[intCount++] = Value;
                                    }
                                }
                            }
                        }
#pragma omp parallel for private(Value)
                        for (int z = 0; z < 3; z++)
                        {
                            for (int j = h2; j < H; j++)
                            {
                                for (int i = 0; i < w2; i++)
                                {
Value = static_cast<short>(round(ImgData[z][i][j]));
                                    if ((Value >= -127) && (Value <= 127))
                                    {
Ret[lPos++] = static_cast<unsigned char>(Value + 127);
                                    }
                                    else
                                    {
Ret[lPos++] = static_cast<unsigned char>(255);
shorts[intCount++] = static_cast<unsigned char>(Value);
                                    }
                                }
                            }
                        }
                    }
                    int shortArraySize = intCount * 2;
```

```
            Array::Resize(Ret, Ret->Length + shortArraySize);
            Buffer::BlockCopy(shorts, 0, Ret, Ret->Length - shortArraySize, shortArraySize);
            pin_ptr<unsigned char> pUnmanagedArr = &Ret[0];
            return pUnmanagedArr;
        }

        double ***Algorithms::WaveletePack(double*** ImgArray, int Component, int cW, int cH, int dwDevider, int dwTop, int dwStep)
        {
            int WV_TOP_TO_BOTTOM=1;
            int WV_LEFT_TO_RIGHT=0;
            short Value;
            int cw2 = cW / 2;
            int cH2 = cH / 2;
            double dbDiv = 1.0f / dwDevider;
            ImgArray = Wv(ImgArray, cW, cH, Component, WV_TOP_TO_BOTTOM);
            ImgArray = Wv(ImgArray, cH, cW, Component, WV_LEFT_TO_RIGHT);
#pragma omp parallel for
            for (int j = 0; j < cH; j++)
            {
                for (int i = 0; i < cW; i++)
                {
                    if ((i >= cw2) || (j >= cH2))
                    {
Value = static_cast<short>(Math::Round(ImgArray[Component][i][j]));
                        if (Value != 0)
                        {
```

```
                                        int value2 = Value;
                                        if (value2 < 0)
                                        {
                                                value2 = -value2;
                                        }
                                        if (value2 < dwTop)
                                        {
                                                ImgArray[Component][i][j] = 0;
                                        }
                                        else
                                        {
                                ImgArray[Component][i][j] = Value * dbDiv;
                                        }
                                }
                        }
                }
        }
        return ImgArray;
}
double ***Algorithms::Wv(double*** ImgArray, int n, int dwCh, int Component, int Side)
{
        int WV_TOP_TO_BOTTOM=1;
        int WV_LEFT_TO_RIGHT=0;
        double a;
        int i, j, n2 = n / 2;
        double* xWavelet = new double[n];
        double* tempbank = new double[n];
        double value = 0;
        for (int dwPos = 0; dwPos < dwCh; dwPos++)
```

```
            {
                if (Side == WV_LEFT_TO_RIGHT)
                {
#pragma omp parallel for //private(value)
                    for (j = 0; j < n; j++)
                    {
                        xWavelet[j] = ImgArray[Component][dwPos][j];
                    }
                }
                else if (Side == WV_TOP_TO_BOTTOM)
                {
#pragma omp parallel for
                    for (int i = 0; i < n; i++)
                    {
                        xWavelet[i] = ImgArray[Component][i][dwPos];
                    }
                }
                a = -1.586134342f;
                for (i = 1; i < n - 1; i += 2)
                {
                xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
                }
                xWavelet[n - 1] += 2 * a * xWavelet[n - 2];
                a = -0.05298011854f;
//#pragma omp parallel for //private(value)
                for (i = 2; i < n; i += 2)
                {
                xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
                }
                xWavelet[0] += 2 * a * xWavelet[1];
```

```
                    a = 0.8829110762f;
                    for (i = 1; i < n - 1; i += 2)
                    {
                            xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i +
1]);
                    }
                    xWavelet[n - 1] += 2 * a * xWavelet[n - 2];
                    a = 0.4435068522f;
                    for (i = 2; i < n; i += 2)
                    {
          xWavelet[i] += a * (xWavelet[i - 1] + xWavelet[i + 1]);
                    }
                    xWavelet[0] += 2 * a * xWavelet[1];
                    a = 1.0f / 1.149604398f;
                    j = 0;
                    if (Side == WV_LEFT_TO_RIGHT)
                    {
                            for (i = 0; i < n2; i++)
                            {
ImgArray[Component][dwPos][i] = xWavelet[j++] / a;
ImgArray[Component][dwPos][n2 + i] = xWavelet[j++] * a;
                            }
                    }
                    else if (Side == WV_TOP_TO_BOTTOM)
                    {
                            for (i = 0; i < n2; i++)
                            {
ImgArray[Component][i][dwPos] = xWavelet[j++] / a;
ImgArray[Component][n2 + i][dwPos] = xWavelet[j++] * a;
                            }
```

```
            }

        }
        return ImgArray;
    }

    double ***Algorithms::YCrCbEncode(unsigned char*** BytesRGB, int cW, int cH, double Ydiv, double Udiv, double Vdiv, int oW, int oH)
    {
        double vr, vg, vb;
        double kr = 0.299, kg = 0.587, kb = 0.114, kr1 = -0.1687, kg1 = 0.3313, kb1 = 0.5, kr2 = 0.5, kg2 = 0.4187, kb2 = 0.0813;
        Ydiv = Ydiv / 100.0f;
        Udiv = Udiv / 100.0f;
        Vdiv = Vdiv / 100.0f;
        double***YCrCb = new double**[3]; //new double[3][cW][cH];
        for (int a = 0; a < 3; a++)
        {
            YCrCb[a] = new double*[cW];
            for (int k = 0; k < cW; k++)
            {
                YCrCb[a][k] = new double[cH];
            }
        }

        for (int j = 0; j < oH; j++)
        {
            for (int i = 0; i < oW; i++)
            {
                vb = static_cast<double>(BytesRGB[0][i][j]);
```

```
                    vg = static_cast<double>(BytesRGB[1][i][j]);
                    vr = static_cast<double>(BytesRGB[2][i][j]);
                    YCrCb[2][i][j] = (kr * vr + kg * vg + kb * vb) * Ydiv;
                    YCrCb[1][i][j] = (kr1 * vr - kg1 * vg + kb1 * vb +
128) * Udiv;
                    YCrCb[0][i][j] = (kr2 * vr - kg2 * vg - kb2 * vb + 128)
* Udiv;
                }
            }
            return YCrCb;
        }
```