Multi-Threading и Hyper-Threading technologies

Multithreading технологии (MT)

MT имат хардуерна поддръжка за ефективно изпълнение на множество нишки. MT се отличава от многопроцесорните системи(както и много-ядрените системи) в това, че нишките трябва да споделят ресурси на единично ядро: изчислителни единици , кеша на CPU и TLB(Translation lookaside buffer). Мултипроцесорните системи включват множество цялостни процесорни блокове. Многонишковостта цели да увеличи използването на единично ядро с използването на нишки както и паралелизъм на ниво инструкции . тъй като двете технологии са взаимно допълващи се, те понякога се съчетават в системи с многожество многонижкови процесори и в процесори с множество многонишкови ядра.

Видове Multithreading:

1.Block Multi-threading

Най-простият вид многонишковост, случва се кога една нишка работи докато е блокирана от събитие, което нормално би създало голяма загуба на латентност. Такова забавяне може да бъде пропуска на кеша, които трябва да достъпи извън-чиповата памет, което може да вземе стотици процесорни цикли за да върне данните. Вместо да чака да се разреши забавянето, нишковият процесор ще превключи изпълнението към друга нишка, която е готова да стартира. Само когато данните за предишната нишка пристигнат, предишната нишка ще бъде поставена обратно на списъка с готовите за стартирани нишки.

Block Multi-threading

Пример:

* 1. Цикъл i : инструкция j от нишка A е издадена
* 2. Цикъл i +1 : инструкция j+1 от нишка A е издадена
* 3. Цикъл i+2 : инструкция j+2 от нишка A е издадена,зарежда инструкция, която липсва в кеша
* 4. Цикъл i+3: срещнат е планирувач на нишка, преминава към нишка B
* 5. Цикъл i+4 : инструкция k от нишка B е издадена
* 6. Цикъл i+5 : инструкция k+1 от нишка B е издадена

Block Multi-threading

Хардуерна цена за реализация:

Целта на хардуера за поддръжка на многонишковост е да позволи бърза смяна на блокирана нишка с друга, готова за изпълнение. За да се осъществи тази цел, хардуерната цена е да възпроизведе програмата видими регистри както и някой контролни регистри (като на пример програмния брояч). Преминаването от една нишка към друга означава хардуера да превключва от използването на един регистър в друг.

Такъв хардуер има следните предимства:

1. Смяната на нишката може да бъде извършено в един CPU цикъл
2. Появява се при всяка нишка, която е изпълнявана сама и не споделя никакви ресурси с други нишки. Това минимизира нужният брой софтуерни промени в приложението както и операционната система да поддържа многонишковост.

За да се превключва ефективно между активни нишки, всяка активна нишка трябва да има собствен наборов регистър. Например, за бързата смяна между две нишки, регистровият хардуер трябва да бъде инициализиран два пъти.

Видове Multi-Threading:

2.Смесена многонижковост (Interleaved/Temporal Multithreading)

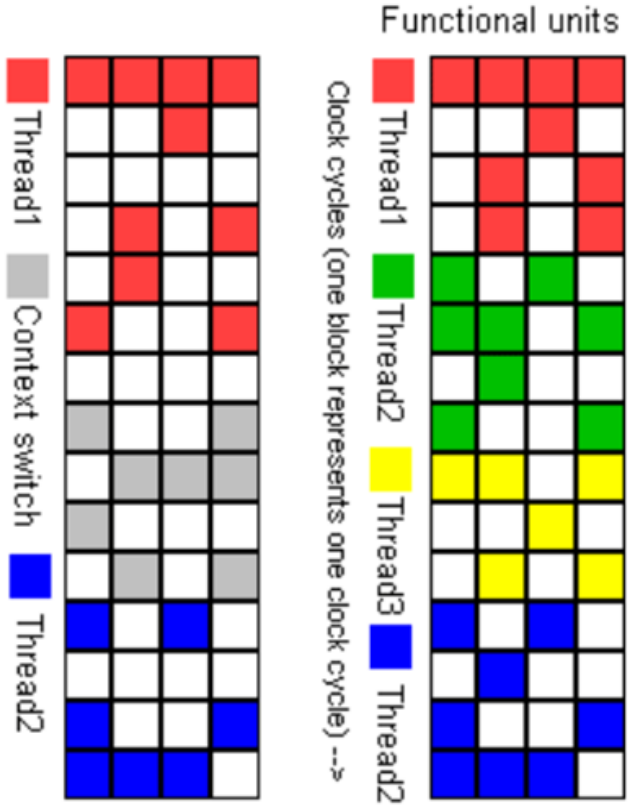
Целта на този тип многонижковост е да се премахнат всичките забавяния от зависимостта от данните от изпълнителния тръбопровод. Тъй като една нишка е относително независима от други нишки, има по-малък шанс една инструкция в една фаза на тръбата да се нуждае от по-стара инструкция в тръбопровода.

Концептуално, този подход е подобен на многозадачният, използван в операционните системи. Може да се направи аналогията , частицата време давана на всяка активна нишка е един CPU цикъл.

Смесената многонижковост е една от двете основни форми на многонижковостта, които могат да бъдат имплементирани на компютърния хардуерен процесор, другата форма е едновременната многонижковост. Отличителната разлика между двете форми е максималният брой конкурентни нишки, които могат да бъдат изпълнени във всяка дадена фаза на тръбопровода в даден цикъл. При смесената многонишковост броят е 1, докато в едновременната е по-голям от 1.

Има много варианти на смесена многонишковост, но повечето могат да бъдат класифицирани в две под-форми: дребно-зърнеста, грубо-зърнеста.

Грубо-зърнеста (Coarse-grained multithreading –CMT)



'a' represents the traditional superscalar, while 'b' represents a coarse-grained multithreading architecture.

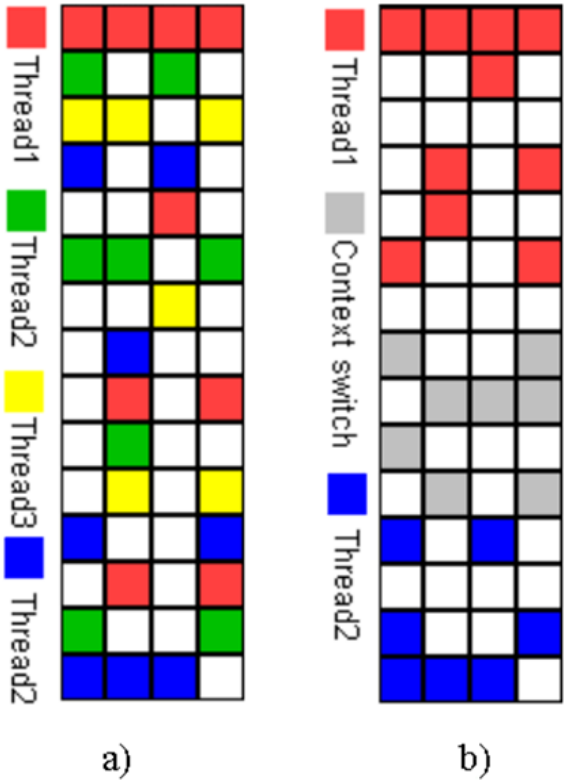
Грубо-зърнеста (Coarse-grained multithreading –CMT)

При грубозърнестата смесена многонижковост, главния процесорен тръбопровод съдържа само една нишка в даден момент. Процесорът трябва ефективно да извърши бърза смяна на съдържанието преди да изпълни друга нишка. Може да има или да няма допълнителни наказателни цикли при смяна.

При преминаване към друга нишка, процесорът запазва състоянието на нишката и преминава към друга. Това става чрез използването на множество набори регистри. Предимството на това се дължи на факта, че често нишка може само да отиде за толкова дълго време преди да попадне на загуба на кеша, или се изчерпи от независими инструкции за изпълнение. CMT процесор може да изпълни само толкова различни нишки по този начин за колкото има поддръжка. Така че може да съхранява само толкова нишки, за колкото има физични локации, за да им съхрани състоянието на тяхното изпълнение.

Има много възможни варианти на едрозърнестта многонишковост, предимно засягащи алгоритъм, които определя кога се извършва смяната. Алгоритъмът може да се основава на един или повече от много различни фактори, включително брой цикли, пропуски на кеша и точност.

Дребнозърнеста(Fine-Grained multithreading - FMT)



'a' is a 4-way fine-grained multithreading processor, and 'b' is a traditional superscalar.

Дребно-зърнеста(Fine-Grained multithreading - FMT)

При Дребнозърнестата временна многонишковост, главния процесорен тръбопровод може да съдържа множество нишки, с ефективно контекстно сменяне, осъществяващо се през фазите на тръбата. Формата на многонижковост може да бъде много по-скъпа от едрозърнестата, защото изпълнимите ресурси които обхващат множество тръбни фази, може да трябва да се справят с множество нишки. Също така, допринася към цената и факта, че този дизайн не може да бъде оптимизиран около концепцията за „фонова“ нишка- всяка от едновременните нишки, имплементирана от хардуера, може да изисква да чете или пише състоянието на всеки цикъл

Смесена многонишковост

* **Хардуерна цена :**

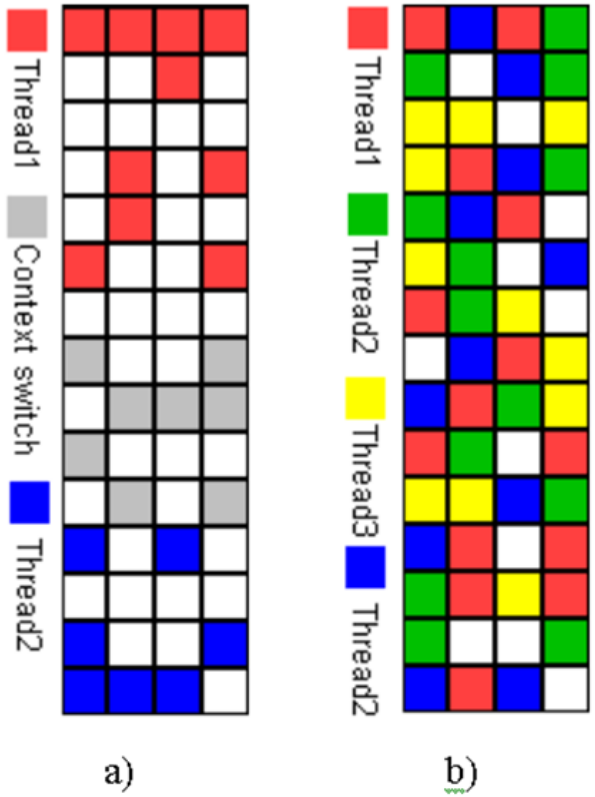
В допълнение на хардуерните разходи обсъдени в Блоковата многонишковост, смесената многонишковост има допълнителни разходи от всеки етап на тръбопровода, следят ID на нишката на инструкцията, която обработва. Също така, тъй като има повече едновременно изпълнявани нишки в тръбопровода, споделените ресурси като кеш и TLBs трябва да се по-големи, за да се избегне „бой“ между различните нишки.

Видове Multi-Threading:

3.Едновременна многонишковост (Simultaneous Multithreading -SMT)

Това е техника за подобряването на цялата ефикасност на супер скаларните процесори с хардуерна многонишковост. SMT позволява множество независими нишки за изпълнение за по-добра употреба на ресурсите, осигурени от съвременните процесорни архитектури.

Едновременна многонишковост



'a' is a traditional super scalar, while 'b' represents a Simultaneous Multithreading architecture.

Едновременна многонишковост

При SMT инструкциите повече от една нишка могат да бъдат изпълнени във всяка дадена фаза на тръбата по всяко време. Това се извършва без големи промени на основната процесорна архитектура: необходимите допълнения са способността да се извлекат инструкции от множество нишки в цикъл и по-голям регистров файл да съдържа данните от множеството нишки. Броят на едновременните нишки може да бъде определен от дизайнерите на чипа, но ограниченията от сложността на чипа, ограничават броят до 2 за повече SMT имплементации, макар че има 8 нишки за ядро-UltraSPARC T2.

Едновременна многонишковост

Тъй като техниката е наистина ефективно решение и има неизбежно увеличен конфликт на споделени ресурси, измерването или съгласуването на ефективността на решението може да бъде трудно. Допълнителните нишки могат да бъдат използвани за активно зареждане на споделен ресурс, като кеш, да подобрят производителността на друга единична нишка. SMT се използва за осигуряване на допълнителни изчисления за някои нива за откриване на грешка и възстановяване.

Въпреки това, в най-актуалните случаи, SMT се използва да скриване на латентността на паметта, повишаване на ефикасността и повишаване на изчислителната способност на изчисленията за количество използван хардуер.

Едновременна многонишковост

Недостатъци:

Едновременната многонишковост не може да подобри производителността, ако някой от споделените ресурси са ограничаващи пречки за производителността. В действителност, някой приложения работят по-бавно, когато е налична едновременна многонишковост. Софтуерните разработчици, трябва да проверят дали едновременната многонишковост е добре или лошо за прилагането и в различни ситуации и допълнителни усилия да се изключи, ако намалява производителността. Сегашните операционни системи не разполагат с удобни API извиквания за тази цел и за предотвратяване на процеси с различен приоритет от взимането на ресурси един от друг.

Multithreading технологии

Предимства :

1) Ако нишка получи много кешови пропуски, друга нишка(и) могат да продължат, възползвайки се от неупотребени изчислителни ресурси, които могат да доведат то по-бързо цялостно изпълнение, тъй като тези ресурси щяха да бъдат незаети, ако само една нишки беше изпълнена.

2) Ако нишка неможе да използва всички изчислителни ресурси на процесора, понеже инструкциите зависят от резултата на една спрямо друга, стартирането на друга нишка позволява да не остави тези неактивни.

3) Ако няколко нишки работят на еднакъв набор от данни, те могат да си споделят техния кеш. Това води до по-добра употреба на кеша или синхронизация на техните стойности.

Multithreading технологии

Недостатъци:

1) Множество нишки могат да се наместват една в друга когато споделят хардуерни ресурси като кешове или TLBs

2) Времето за изпълнение на единична нишка не е подобрено, но може да бъде разградено, дори когато се изпълнява една нишка. Това се дължи на по-бавните честоти и/или допълнителни pipeline фази, които са необходими за установяване за хардуера за превключване на нишките.

3) Многонишковата хардуерна поддръжка е по-видима при софтуера, по този начин изисква повече промени в приложните програми и в операционните системи отколкото при многопроцесорността.

Hyper-Threading технология

Hyper-Threading технологията на Intel носи концепцията на едновременна многонишковост на Intel архитектурата.Hyper-Threading технологията прави един физичен процесор да изглежда като два логически процесора. Физическите изпълними ресурси са споделени и архитектурната структура се дублира за двата логически процесора. От софтуерна или архитектурна гледна точка, това означава, че операционните системи и потребителските програми могат да планират процеси или нишки към логическите процесори,както биха осъществили и на множество физични процесори. От микроархитектурна гледна точка, това означава, че инструкциите от двата логически процесора ще се запазят и изпълнят едновременно на споделени изпълними ресурси.

Микроархитектура на процесора

Традиционните подходи за дизайн на процесор се фокусират върху по-високи тактови честоти, паралелизъм на ниво инструкции и кеш. Техники за постигане на по-високи тактови честоти включват pipelining на микроархитекрурата до по-финни гранули-super-pipelining. По-високите тактови честоти могат значително да подобрят производителността, като увеличат броят на инструкциите, които могат да бъдат изпълнение всяка секунда. Поради това ще има много повече изпълнявани инструкции в super-pipelined микроархитектурата, обработващи събития, които нарушават „тръбопровода“, например пропуска кеш, прекъсва и не извършва предвиждания на преходите, това може да е скъпо.

Микроархитектура на процесора

ILP(Instruction-level parallelism) се отнася до техники, за да увеличи броят на инструкциите, които се изпълняват всеки такт. Например супер-скаларния процесор има много паралелно изпълними единици, които могат да изпълнят инструкциите едновременно. Със супер-скаларно изпълнение, няколко инструкции могат да бъдат изпълнявани всеки такт. Въпреки това, с просто неспазващо реда изпълнение, не е достатъчно само да има множествено изпълними единици. Предизвикателството е да се намерят достатъчно инструкции за изпълнение. Една техника е out-of-order изпълнение, където голям прозорец от инструкции е едновременно оценяван и изпращан към изпълняващите единици, въз основа повече на инструкционни зависимости, отколкото на програмен ред.

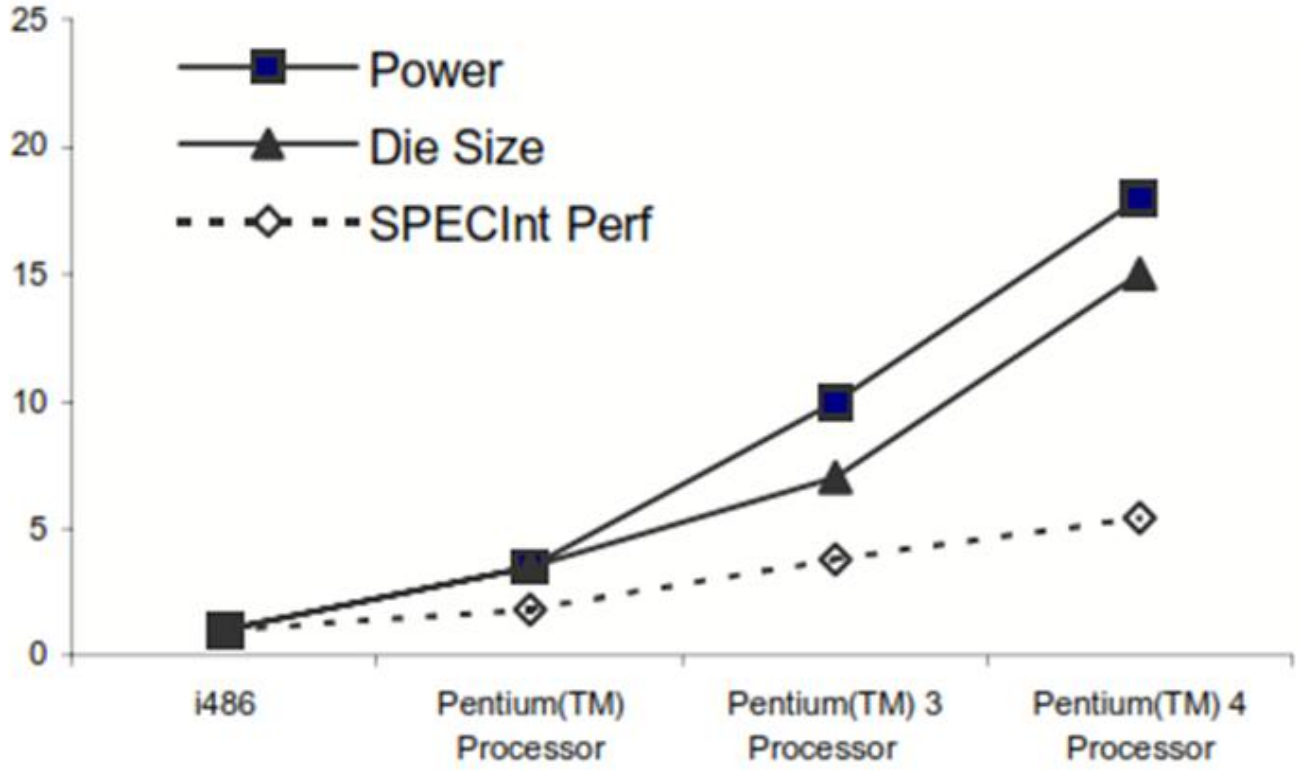
Микроархитектура на процесора

Достъпите до DRAM паметта са бавни в сравнение със скоростта на изпълнение на процесора. Една техника, за да се намали тази латентност е да се добави бърз кеш близо до процесора. Кешът може да осигури бързи достъп до паметта до често достъпни данни или инструкции. Обаче кешът може да е бърз само когато е с малък размер. Поради тази причина, процесорите често са проектирани с кеш йерархия, в която се намират бързи, малки кеш памети, разположени и опериращи с достъп тайминги, много близо до тези на ядрото на процесора, и все по-големи кеш, които се справят с по-малко честотно-достъпни данни или инструкции, изпълнявани с по-дълги тайминги за достъп. Въпреки това, винаги ще има моменти, когато необходимите данни няма да са в никоя процесорна кеш. Справянето с такъв кеш пропуска изискваната достъпващата памет и е вероятно процесорът бързо да бъде лишен от инструкции за изпълнение преди сриването на кеш пропуска.

Микроархитектура на процесора

Все по-голяма част от техниките за подобряване на производителността на процесора от едно поколение към следващо е сложна е често прибавя значително оразмеряване и разходи на мощност. Тези техники увеличават производителността, но не и с 100% ефективност, т.е. удвоявайки броя на изпълнителните единици в един процесор не удвоява производителността на процесора, поради ограничен паралелизъм в инструкционния поток. По същия начин, простото удвояване на тактовата честота не удвоява производителността поради броят на процесорни цикли, загубени поради непредвидени преходи.

Микроархитектура на процесора



Микроархитектура на процесора

Тази графика показва относително нарастване на производителността и разходите,като оразмеряване мощност през последните 10 години на Intel процесорите. Тези данни са приблизителни и са предназначени само да покажат тенденциите, не действителната производителност. За да се изолира въздействието на микроархитектурата, това сравнение приема , че четири поколения процесори са на същия силициево-технологичен процес и че високата скорост е нормализирана към производителността на процесор Intel 486. Въпреки, че използваме историята на процесор Intel в този пример, други производители на високопроизводителен процесор през този период ще имат подобни тенденции. Производителността на Intel процесор, поради микроархитектурния напредък, сам по себе се, като цяло число се е подобрил 5-6 пъти. Повечето целочислени приложения имат ограничен ILP и инструкционния поток трудно може да бъде предсказан.

Микроархитектура на процесора

* През същия период, относителното оразмеряване се е увеличило 15 пъти, 3 пъти по-високо от печалбите при целочислената производителност. За щастие, напредъкът в силициево технологичния процес, позволява повече процесори да бъдат „опаковани“ в дадено количество площ, така че действителната измервана площ на всяко поколение микроархитектура не се е увеличило значително.
* Относителната мощност се увеличава почти 18 пъти през този период. За щастие съществуват редица познати техники за значително намаляване на консумацията на мощност на процесорите и има много продължаващи изследвания в тази област. Въпреки това, сегашната разсейвана процесорна мощност е на границата на това,с което лесно може да се справят десктоп платформи и трябва да поставим по-голям акцент на подобряване на работата във връзка с нова технология, която специално да контролира мощността.

Паралелизъм на ниво нишки:

Поглед към съвременните софтуерни тенденции показва, че сървър приложенията се състоят от множество нишки или процеси, който могат да бъдат изпълнено паралелно. Онлайн изпълнението на транзакции и Web услуги имат изобилие от софтуерни нишки, които могат да бъдат изпълнени едновременно за по-бързо изпълнение на задачите. Дори и настолните приложения стават все по паралелни. Архитектите на Intel се опитват да увеличат така наречения паралелизъм на ниво нишки(TLP), за да постигнат по-добра производителност в сравнение със съотношението на броя на транзисторите и мощността

Паралелизъм на ниво нишки:

Както във високият, така и в средният клас сървърни пазари, мултипроцесорите са често използвани, за да постигне по-голяма производителност от системата. Чрез добавяне на повече процесори, приложенията имат потенциала да постигнат съществени подобрения върху производителността, като изпълняват множествено нишки на множество процесори в едно и също време. Тези нишки може да са от едно и също приложение, от различни приложения изпълнявани едновременно, от сървиси на операционната система, или от нишки на операционната система извършващи поддръжка зад фона. Мултипроцесорните системи са били използвани в продължение на много години, и висококласните програмисти са запознати с техниките да използват мултипроцесорите за по-високи нива на производителност.

Паралелизъм на ниво нишки:

През последните години броят на други техники за по-нататъшно използване на TLP биват обсъждани и някои продукти са обявени. Една от тази техники е чип multiprocessing (CMP), където два процесора са поставени на единичен блок. Двата процесора, всеки от които има пълен набор от изпълнителни и архитектурни ресурси. Процесорите могат или не могат да споделят голяма вградена памет. CMP е до голяма степен ортогонален д към конвенционални микропроцесорни системи, тъй като може да имаме много CMP процесора в една мултипроцесорна конфигурация. CMP чипа е значително по-голям от размера на едноядрен чип и следователно е по-скъп за производство, освен това той не се справя с размера на блокът и разсеяната мощност.

Паралелизъм на ниво нишки:

Друг подход е да се позволи на единичен процесор да изпълнява множество нишки, като ги превключва. Времевият отрязък на многонишковост може да доведе до пропиляване на изпълними на слотове, но може ефективно да сведе до минимум въздействието на дългия тайминги на паметта. Switch-on-event многонишковост ще превключи нишки без дълготрайни събития като кеш пропуски. Този подход може да работи добре за сървърни приложения, които имат голям брой кеш пропуски и където две нишки изпълняват подобни задачи. Въпреки това, и двете многонишкови техники – времевия отрязък превключване на събитие не постигат оптимално препокриване на много източници на неефективно използване на ресурсите, като непредвиждания на преходите, инструкционни зависимости и др.

Паралелизъм на ниво нишки:

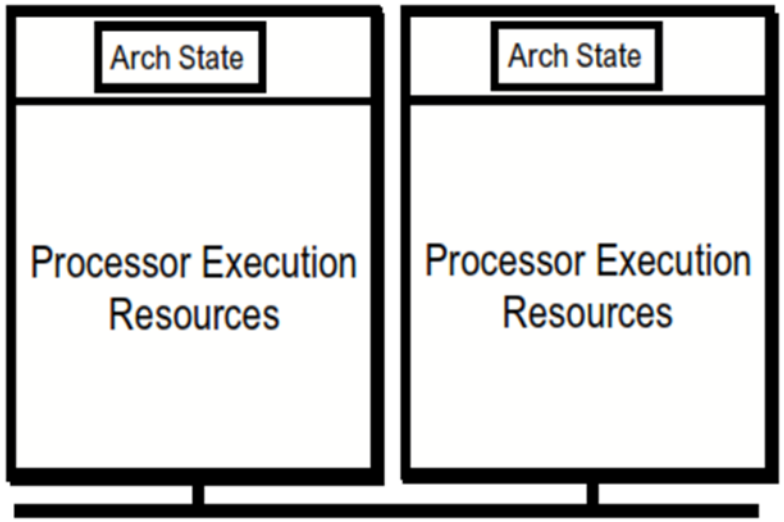
Съществува едновременна многонишковост, където множество нишки могат да бъдат изпълнени на единичен процесор без да се превключват. Нишките се изпълняват едновременно и правят много по-добра употребата на ресурсите. Този подход прави най ефективната употребата на процесорни ресурси. Той увеличава производителността в сравнение с брои транзистори и консумация н енергия.

Hyper-Threading технологията допринася едновременния многонишков подход на Intel архитектурата.

Архитектура на Hyper-Threading технологията :

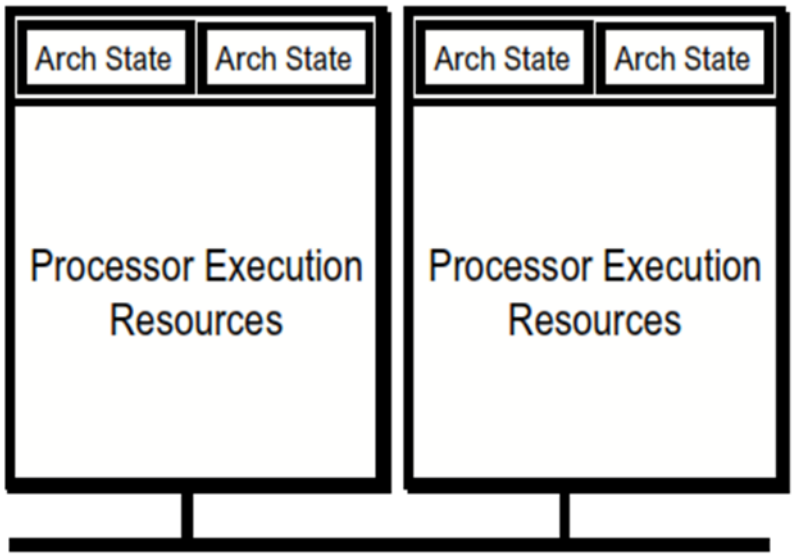
Hyper-Threading технологията прави един физически процесор да изглежда като множество логически процесори. За да направи това, има едно копие на архитектурното състояние за всеки логически процесор и логическите процесори споделят единичен набор от физични изпълними ресурси. От гледна точна точка на софтуера или архитектурата , това означава, че операционната система и потребителските програми могат да планират процеси или нишки към логическите процесори, както биха направили на конвенционален физически процесор в мултипроцесорна система. От микроархитектура гледна точка, това означава, че инструкциите от логическите процесори ще съществуват и изпълняват едновременно на споделени ресурси за изпълнение.

Архитектура на Hyper-Threading технологията :



Мултипроцесорна система с два процесора, които не са Hyper-Threading съвместими.

Архитектура на Hyper-Threading технологията :



Мултипроцесорна система с 2 физични процесора, които са Hyper Threading съвместими. С две копия на архитектурното състояния на всеки физичен процесор. Системата има 4 логически процесора.

Архитектура на Hyper-Threading технологията :

* Първата имплементация на Hyper-Threading технологията е направена възможна на процесори от семейството на Intel Xeon за дву- и многопроцесорни сървъри с два логически процесора за физичен процесор. Чрез по-ефективно използване на съществуващите процесорни ресурси, семейството на Intel Xeon могат значително да подобрят производителността в почти същите загуби на системата. Това внедряване на Hyper-Threading технологията добавя по-малко от 5% на размера на чипа и максималните изисквания за мощност, но може да осигури печалба в производството много повече от това.

Архитектура на Hyper-Threading технологията :

Всеки логичен процесор поддържа пълен набор от архитектурно състояние. Архитектурното състояние се състои от регистри, включващи регистри с общо предназначение, контролни регистри, подробните програриуеми прекъсващи регистри (APIC), и някой регистри за машинните състояния. От софтуерна гледна точка, след като се дублира архитертурното състояние, процесорът изглежда да е два процесора. Броят на транзистори за съхранение на архитектурното състояние е изключително малка част от общия брой. Логическите процесори споделят почти всички други ресурси на физичния процесор, като кеш, изпълними единици, branch predictors и шини.

Всеки логичен процесор има негов собствен прекъсващ контролер. Прекъсванията изпратени до определен логичен процесор са обработвани само от този логичен процесор.

Първо внедряване на процесори от Xeon фамилията

Няколко цели са били в сърцето на микроархитектурните проектни решения, направени за процесор Intel Xeon процесор,мултипроцесор изпълнение на Hyper-Threading технологията. Една от целите е да се сведат до структурната площ за имплементираните на Hyper-Threading технологията. Тъй като логическите процесори споделят по-голямата част на микроархитектурните ресурси и само няколко малки структури са повтарят, разходите за площта на първото изпълнение били по-малки от 5% от общата площ.

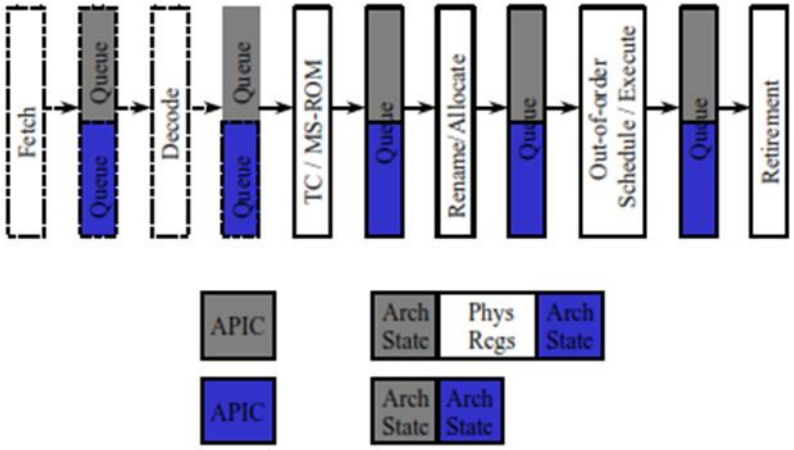
Първо внедряване на процесори от Xeon фамилията

Втората цел е да се гарантира, че когато един логически процесор е в застой, другия логически процесор може да продължи да действа нататък. Логическия процесор може да бъде временно в застой за най-различни причини, включително обслужване на кешови пропуски,непредсказване на преходите или чакащи за резултата от предишни инструкции. Независимото действие нататък е осигурено от управлението на буферните опашки, така че никой логически процесор да може да използва всички записи, когато се изпълняват две активни софтуерни нишки. Това се постига от входно разделяне или ограничаване на броят на активните записи за всяка нишка.

Първо внедряване на процесори от Xeon фамилията

Третата цел е да позволи процесор, работещ само една активна софтуерна нишка, да работи със същата скорост на процесор с Hyper Threading технология, като на процесор без тази технология. Това означава, че разделените ресурси трябва да се възстановят, когато само една нишка е активна. Изглед на високо ниво с оглед на микроархитектурния тръбопровод е показан на фигурата:

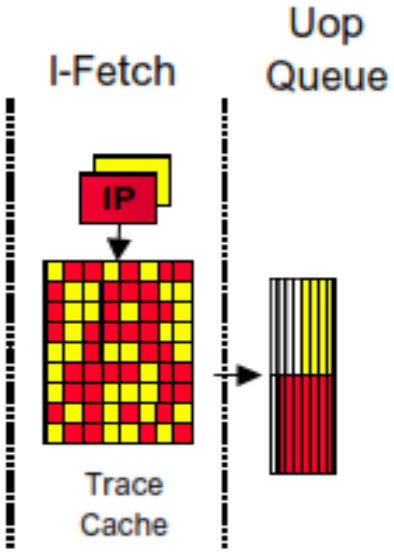
Първо внедряване на процесори от Xeon фамилията



Буферираните опашки отделят съществени логически тръбопроводни блокове. Те са или разделени или дублирани, за да осигурят независимо продължение напред чрез всеки логически блок.

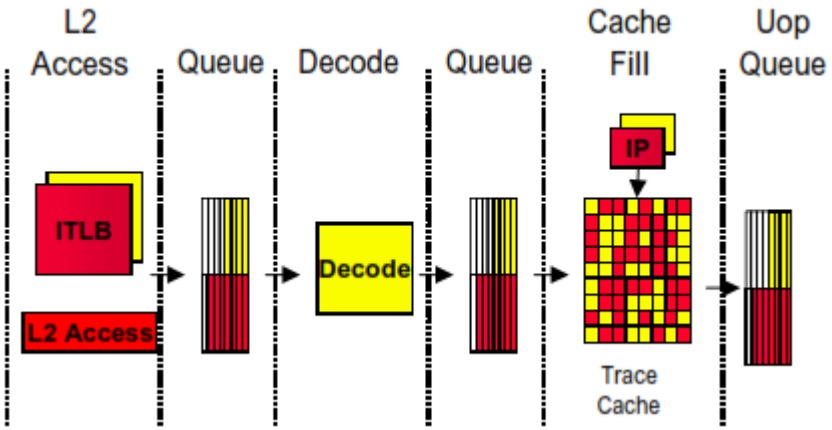
Front End:

* Предният край на тръбопровода е отговорен за доставянето на инструкции за по-късните тръбни етапи.



Инструкциите идват от Execution Trace Cache(TC), който е основният кеш или L1 инструкционен кеш.

Front End:



При този случай, само когато липса TC, машината ще извлече и декодира инструкции от вградения кеш –L2. Близко до TC се намира Microcode ROM, който съхранява разкодираните инструкции за по-продължително и по-сложни IA-32 инструкции.

Execution Trace Cache(TC)

Той съхранява декодираните инструкции, наречени микрооперации . Повечето инструкции в програмата се извличат и изпълняват от TC. Два набора от указатели на следващи инструкции независимо следят прогреса на изпълнение на двете софтуерни нишки. Двата логически процесора определят достъпа до TC на всеки оперативен такт. Ако двата логически процесора искат достъп до TC в едно и също време, достъпът е предоставян до единият после до другия в редуващ тактов цикъл. Например, ако единият цикъл е използван да извлече линия за единия логически процесор, другият цикъл ще бъде използван да извлече за другият логически процесор, осигурявайки на двата логически процесора поискания достъп до trace cache. Ако единият логически процесор е в застой или е неспособен да използва TC, другият логически процесор при всеки цикъл може да използва пълния размер на trace cache . TC записите са маркирани с нишкова информация и динамично се разпределят при нужда. TC е 8 степенен асоциативен набор и записите се заменят въз основа на най-малко скорошно употребявани алгоритъм, който се основава на целите 8 начина. Споделената природа на TC позволява при необходимост единия логически процесор да има повече записи отколкото другия.

Microcode ROM

Когато се срещне сложна инструкция, TC изпраща микро-код указател до Microcode ROM. Контролерът на Microcode ROM тогава извлича нужните микро-код операции у връща контрол към TC. Два микро-код инструкционни указателя се използват да контролират независимо потоците, ако двата логически процесора изпълняват сложни IA-32 инструкции.

Двата логически процесора споделят Microcode ROM записите. Достъпа до Microcode ROM се редува между логическите процесора, също както и в TC.

ITLB u Branch Prediction

Ако има TC пропуск, тогава инструкционните байтове трябва да са извлекат от L2 кешът и декодирани в микро-код операции, за да бъдат поставени в TC. ITLB(Instruction Translation Lookaside Buffer) получава искане от TC да достави нови инструкции и транслира указателят към следващия инструкционнен адрес към физически адрес. Искането е изпратено до L2 кешът и са върнати инструкционни байтове. Тези байтове се поставят в потокови буфери, които задържат байтовете докато могат да бъдат декодирани. ITLB се дублират. Всеки логически процесор има свой собствен ITLV и собствен набор от инструкционни указатели да следят прогресът на инструкционното извличане за двата логически процесора. Логиката на инструкционното извличане , отговарящо за изпращане на заявки към L2 кешът, определящ на първото дошло искане да бъде първо обслужено, като винаги запазва най-малко един слот за заявки за всеки логически процесор. По този начин и двата логически процесора могат да имат висящи извличания едновременно.

ITLB u Branch Prediction

Всеки логически процесор има собствен набор от 64-битови потокови буфери, които задържат инструкционните байтове в подготовка за фазата на инструкционно декодиране. ITLBs и потоковите буфери са малки структури, така че цената за дублиране на тези структури е нищожно малка. Структурите на прогнозата за прехода са или дублирани или споделени. Стековият буфер за връщане, който предсказва целта на връщаните инструкции, е дублиран, защото е много малка структура и извикваните/връщани чифтове са по-добре независимо предсказуеми за софтуерните нишки. Буферът за история на преходите се използва да търси, дали масива на глобалната история е също независимо проследен за всеки логичен процесор. Обаче големият глобален масив на историята е споделена структура със записи, който са маркирани с ID на логически процесор.

Декодиране на IA-32 инструкция

Инструкциите IA-32 са тромави за декодиране, защото инструкциите имат променлив брой байтове и имат много различни варианти. Значително количество логика и междинно състояние е необходимо да декодира тези инструкции. За щастие, TC осигурява повечето от микро-кода и декодирането е необходимост само за инструкциите, които пропускат TC.

Декодиране на IA-32 инструкция

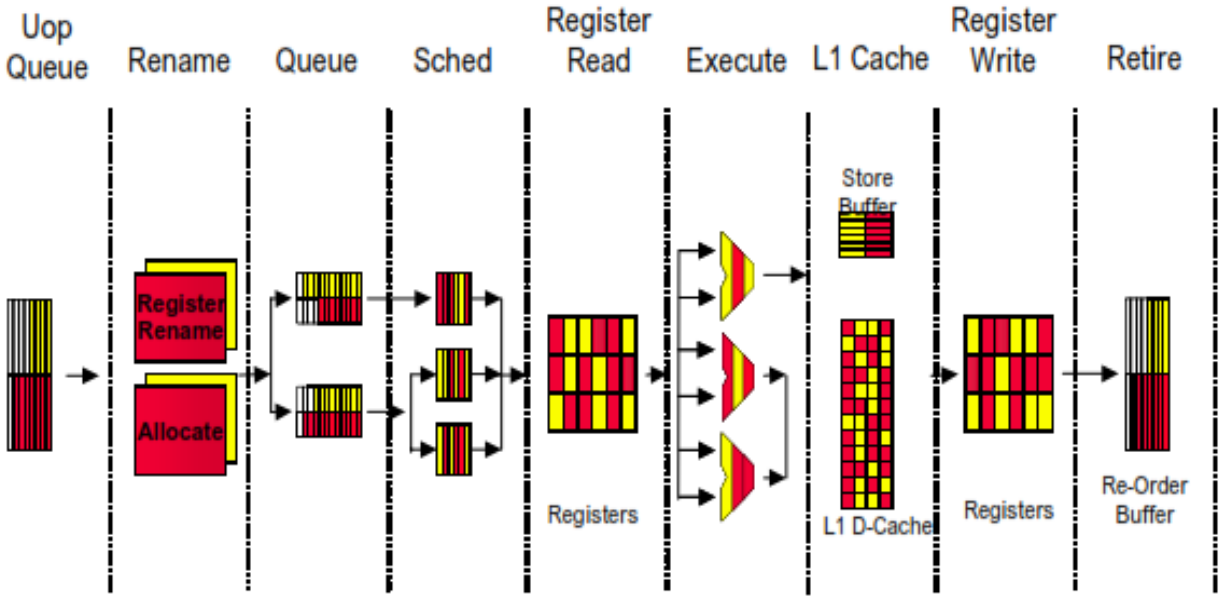
Логиката на декодирането взима инструкционни битове от потоковите буфери и ги декодира в микрокодове. Когато нишките декодират инструкции едновременно, потоковите буфери се превключват между нишките така двете нишки споделят една и съща логика на декодера. Логиката на декодера трябва да пази две копия на всички състояния, нужни да декодират инструкции IA-32 за двата логически процесора, дори и когато се декодират инструкции за един логически процесор по това време. Като цяло, няколко инструкции се декодират за един логически процесор преди преминаването към другият логически процесор. Решението да се направи грубо ниво на гранулярност в превключването между логически процесори е направено в интерес на структурния размер и да се намали сложността. Разбира се, ако само един логически процесор се нуждае от декодираща логика, целия декодиращ размер е отдаден на този логически процесор. Декодираните инструкции се записват в TC и пренасочват към опашката на микрокод операциите.

Опашка на микро-код операции

След като микрокод операциите са извлечени от trace cache или Microcode ROM, или пренасочени от инструкционната декодираща логика, те се поставят в „опашката на микрод операции“. Тази опашка отделя Front End от Out-of-order Execution Engine в тръбния поток. Опашката е разделена като всеки логически процесор има половината записи. Това разделяне позволява двата логически процесора да напредват независимо напред независимо от front-end пропуските или пропуски при изпълнението.

Out-of-order Execution Engine

Състои се разпределение, преименуване на регистри, планирани и изпълнителна функция



Тази част от машината пренарежда инструкциите и ги изпълнява толкова бързо, колко техните входове са готови, без да се съобразява с оригиналния програмен ред

Allocator(Разпределител)

The out-of-order execution engine има няколко буфера за изпълнение на пренареждането, проследяването и последователността на операциите. Разпределителната логика взема микрокод операции от опашката и разпределя много от ключовите машинни буфери, необходими да изпълнят всяка микрокод операция, включително 126 разместващи буферни записа, 128 целочислени и 128 с плаваща запетая физични регистри, 48 зареждащи и 24 записващи буферни записи. Някой от тези ключови буфери са разпределени така, че всеки логически процесор може да използва повече от половината записи.

По-конкретно, всеки логически процесор може да използва до максимум от 63 пренаредени буферни записи, 24 зареждащи буфери и 12 записващи буферни записи.

Allocator(Разпределител)

* Ако има микрокод операции за двата логически процесора в опашката, разпределителят ще редува избирането на микрокодове операции при всяка тактова честота на логическите процесори, за да придобие ресурси. Ако логическият процесор използва неговия лимит на необходими ресурси, например буферни съхраняващи регистри, разпределителят ще сигнализира „закъснение“ за този логически процесор и ще продължи да присвоява ресурси за другия логически процесор. Освен това, ако опашката съдържа само микрокодове за един логически процесор, разпределителят ще пробва да присвои ресурси при всеки такт за този логически процесор, за да оптимизира разпределителната големина, въпреки лимитът на ресурсите все пак ще бъде изпълнен.
* Чрез лимитирането на максималното използване на ресурсите на ключови буфери, машината помага за прилагане на коректност и предотвратява deadlocks.

Register rename(Преименуване на регистъра)

Преименува архитектурните IA32 регистри в машинни физични регистри. Това позволява 8-те IA-32 целочислени регистри за общо ползване да бъдат динамично разширени да използват наличните 128 физични регистъра. Преименуващата логика използва Register Alias Table(RAT) да проследи най-последната версия на всеки архитектурен регистър, да му каже следващата инструкции(я), къде да получи неговите входни операнди.

Тъй като всеки логически процесор трябва да поддържа и следи собственото си пълно архитектурно състояние, има две RAT, по една за всеки логически процесор. Процесът по преименуване на регистъра е извършван паралелно на разпределителят, така логиката на преименуване на регистъра работи на същите микрокод операции, на които разпределителят присвоява ресурси.

Register rename(Преименуване на регистъра)

След като микрокод операции са завършили разпределението и процесите по преименуване на регистрите, те се поставят в два набора от опашки, един за операции свързани с паметта(зареждане и съхраняване) и друг за всички други операции. Двата набора от опашки се наричат memory instruction queue(опашка за паметни инструкции) и съответно general instruction queue(опашка за главни инструкции). Двата набора опашки са също разделение, така че микрокод операции от един логически процесор може да използва повече от половината записи.

Instruction Scheduling(Планиране на инструкциите)

Планирането е в сърцето на out-of-order execution engine. Пет микрокод операции „планирувачи“ се използват да планират различни видове микрокод операции за различните изпълними единици. Заедно те могат да изпращат до 6 микрокод операции на всеки тактов цикъл. Планирувачите определят кога микрокод операции е готова да бъде изпълнена въз основа на готовността на техните зависими входни регистрови операнди и способността на ресурсите на изпълнимата единица.

Инструкционната опашка на паметта и общите инструкционни опашки изпращат микрокод операции до петте планиращи опашки колко се може по-бързо, превключвайки между микрокод операции за двата логически процесор на всеки тактов цикъл, колкото е необходимо.

Instruction Scheduling(Планиране на инструкциите)

Всеки „планирувач“ има собствена планираща опашка от 8 до 12 записа от които той избира микрокод операции да изпрати до изпълнимите единици. Планирувачите избират микрокод операции според това, дали те принадлежат на единия логически процесор или на другия. Планирувача ефективно се съобразява с характеристиките на логически процесор. Микрокод операции са оценяват, базирайки се зависимите входове и способността на изпълнимите ресурси. Например, планирувача може да изпраща 2 микрокод операции от един логически процесор и 2 микрокод операции от другия логически процесор в един и същ тактов цикъл. За да се избегне deadlock и осигури коректност има лимит на броят на активни записи, които логически процесор може да има във всяла планираща опашка. Този лимит зависи от размерът на планиращата опашка.

Execution Units(Изпълними единици

Изпълнимото ядро и йерархията на паметта също до голяма степен се съобразяват с логически процесор. Тъй като източникът и отправните регистри бяха преименувани по-рано до физични регистри, микрокод операции просто достъпва физичния регистров файл, за да вземе техните дестинации, и те записват резултата обратно на физичния регистров файл. Сравнявайки физични регистрови номера, позволява пренасочващата логика да препрати резултати до други изпълними микрокод операции без да трябва да разберат логически процесор.

След изпълнението, микрокод операции се поставят в разместващия буфер. Разместващия буфер разделя изпълнимата фаза от оттеглящата. Разместващият буфер е разделен така, че всеки логически процесор може да използва половината записи.

Retirement (Оттегляне)

Логиката на оттеглянето на микрокод операции се поверява на архитектурното състояние в програмния ред. Оттеглящата логика следи кога микрокод операции от 2 логически процесора са готови да бъдат оттегляни, тогава оттегля микрокод операции в програмен ред за всеки логически процесор като се редува между двата логически процесора. Оттеглящата логика ще оттегли микрокод операции за един логически процесор, тогава другия, редуващи се назад и напред. Ако един логически процесор не е готов да оттегли никоя микрокод операции тогава всички оттеглящ размер е предоставен на другият логически процесор.

След като записите се оттеглят, съхранените данни трябва да бъдат записани в L1 cache. Изборна логика превключва между двата логически процесор да се заемат да съхранят данните на кешът.

Memory Subsystem(Подсистема памет)

Подсистема памет включва DTLB, ниско латентен кеш за данни L1, обединен кеш L2, и обединен кеш L3(наличен е само при Intel Xeon мултипроцесора). Достъпът до подсистемната памет също се съобразява с логически процесор. Планирувачите изпраща заредени или съхранени микрокод операции без оглед на логически процесор и когато пристигат, подсистемната памет се заема с тях.

DTLB
(Dual Translation Lookaside Buffer)

Превежда адреси до физически адреси. Има 64 напълно асоциативни записи. Всеки запис може да изобрази 4К или 4МБ страница. Въпреки че DTLB е споделена структура между два логически процесор, всеки запис включва ID маркер на логически процесор. Всеки логически процесор има запазен регистър, който са осигури качество и да нататъшен процес в обработката на пропуските на DTLB.

Кеш за данни L1,кеш L2, кеш L3

Кешът за данни L1 е 4-лентов асоциативен с 64 битови линии. Осъществява се запис през кеша, т.е. винаги се копира в L2 кеша. Кеш за данни L1 е виртуално адресиран и физично маркиран. Кешовете L2 и L3 са 8-лентови асоциативни с 128 битови линии. L2 и L3 кешовете са физично адресирани. Двата логически процесор без оглед на микрокод операции на логически процесор може първоначално да са подали данните в кешът, могат да споделят всички записи във всичките 3 нива на кеша.

Кеш за данни L1,кеш L2, кеш L3

Тъй като логическия процесор могат да споделят данни в кеша, могат да възникнат конфликти, които да доведат до по-ниска производителност. Обаче има възможност за споделяне на данни в кеша. Например , един логически процесор може да преизвлече инструкции или данни, нужни на другия логически процесор, в кеша. Това е често срещано в сървърните приложения. В потребителско насочените модели, един логически процесор може да произведе данни, които другият логически процесор иска да използва. В такива случаи има потенциал за добри производителни ползи.

Bus(Шина)

* Заявките на паметта на логически процесор не са удовлетворени от йерархията на кеша и са обслужват от логиката на шината. Тя включва локален APIC контролер на прекъсванията, който е извън чипа на системната памет и I/O пространството. Шинната логика се занимава и с кешираща адресна кохерентност на заявките, произхождащи от други външни шинни агенти, както и доставка на входящи заявки за прекъсвания чрез локалните APIC.
* От друга гледна точка, заявките от логически процесор се третират на принципа пръв пристигнал, с опашка и буферно споделено пространство. Приоритетът не се дава на един логически процесор пред другия.

Bus(Шина)

* Разграниченията между исканията от логически процесор са надеждно поддържани в шинните опашки. Заявки към локалните APIC и ресурси на доставка на прекъсвания са уникални и отделни за всеки процесор. Шинната логика също носи части на бариерната преграда и операция за нареждане на паметта, които се прилагат на опашките на шинната заявка за основата на логически процесор.
* За отстраняване на грешки и като помощ на механизмите за продължаване напред в клъстерните мултипроцесор имплементации, ID на логически процесор е видимо изпращано на процесорната външна шина в частта на исканата фаза на транзакция. Други шинни транзакции, като избягване на кеш линии или преизвличане на транзакции, наследяват ID на логически процесор на заявката, която генерира транзакцията.

Еднозадачов и многозадачов режим

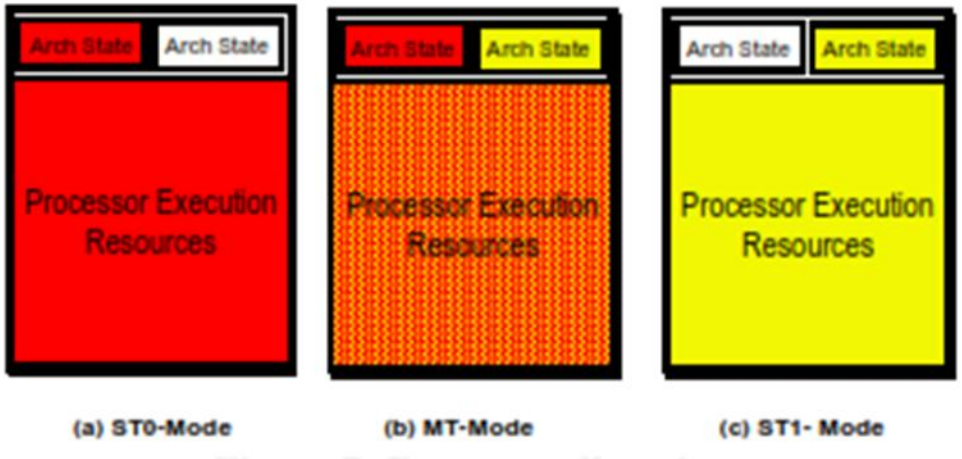
За подобряване на производителността когато има една софтуерна нишка за изпълнение, има два режима на работа, отнасящи се като еднозадачен(ST) или многозадачен(MT) . При MT режим, има два активни логически процесор и някой от ресурсите са разделят, като се описват преди това. Има две разновидности на ST режима : еднозадачен логически процесор 0 (ST0) и еднозадачен логически процесор 1(ST1). При ST1 или ST0 режим, само един логически процесор е активен и ресурсите, които са били разделени в MT режим се рекомбинират да се дадат на единичен активен логически процесор да ги използва всичките. Интелската архитектура IA-32 има инструкция наречена HALT, която спира изпълнението на процесора и позволява процесора да премине в икономичен режим. HALT е привилегирова инструкция, означаваща, че само операционната система или други ring-0 процеси могат да изпълняват тази инструкция. Потребителските приложения не могат да извършват HALT.

Еднозадачов и многозадачов режим

На процесор с HTT изпълнявайки HALT транзакции , процесорът от MT режим до ST0 или ST1 режим, зависи на кои логически процесор е изпълнявана инструкцията HALT. Например ако логически процесор 0 изпълнява HALT, само логически процесор 1 ще бъде активен. Физическият процесор ще бъде в ST1 режим и разделените ресурси ще бъдат рекомбинирани, давайки на логически процесор 1 пълно ползване на всички процесорни ресурси. Ако оставащият активен логически процесор също изпълнява HALT, физичния процесор тогава ще бъде способен да премине в икономичен режим.

Еднозадачов и многозадачов режим

При ST0 и ST1 режими, прекъсване изпратено до процесор, изпълняван HALT, ще доведе транзакция до MT режим. Операционната система е способна за управлението на MT транзакциите.

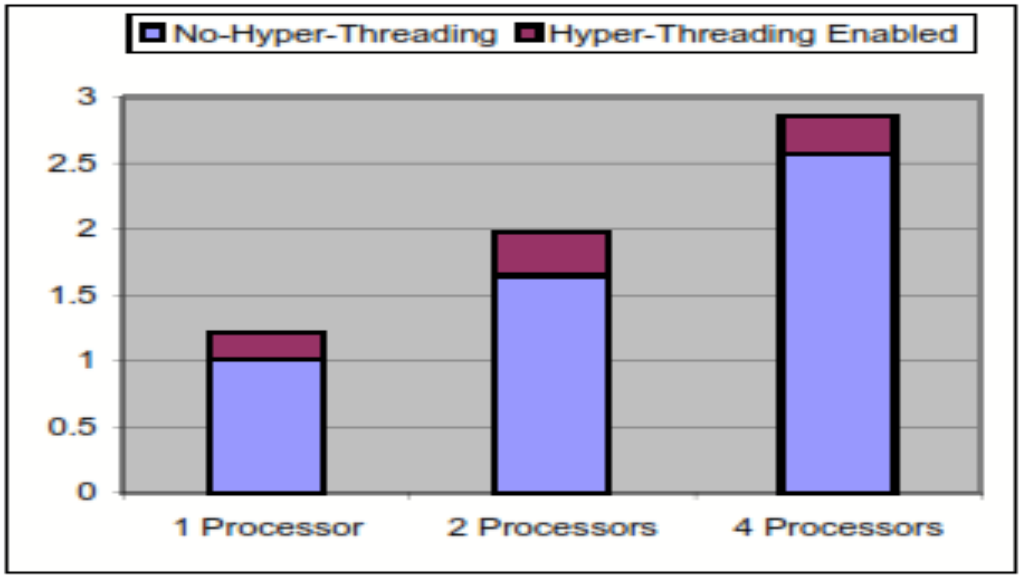


При процесор с HTT, ресурсите са разполагат на единичен логически процесор ако процесорът е в режим ST0 или ST1. При MT режим, ресурсите се споделят между двата логически процесор.

Производителност

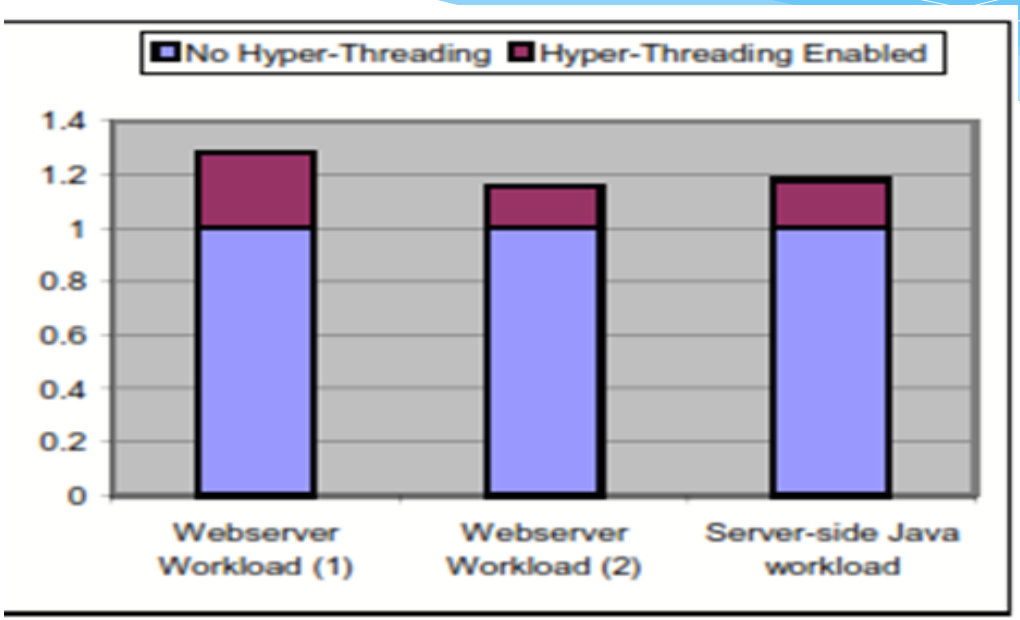
Процесорите от семейството на Intel Xeon осигуряват най-високата системна производителност на всяка Интелска архитектура от тип IA-32. Първоначалните тестове показват до 65% увеличение на производителността на висок клас сървърни приложения, сравнение с предно поколение Pentium III Xeon процесор на 4 степенни сървърни платформи. Значителна част от тези печалби за от HTT.

Производителност



Производителност при извършване на онлайн транзакции, мащабирана от едно-процесорна система до 4-процесорна система с HTT. Има значителна печалба на производителност при HTT, 21% при едно и двупроцесорни системи.

Производителност



Печалбата на HTT при изпълнението на други сървърно насочени изследвания. Печалбата при производителността варира от 16 до 28%.

Развитие на Hyper-Threading технологията

* Hyper-Threading технологията е въведена първо във Foster MP-базирания Xeon през Март, 2002. Той е използван на 3.06 GHz Northwood-базиран Pentium 4, същата година, и тогава е използван във всеки Pentium 4 HT, Pentium 4 Extreme Edition и Pentium Extreme Edition процесори.
* Предишните поколения на интелските процесори, базирани на Core микроархитектурата нямат Hyper-Threading, защото Core архитектурата е потомък на P6 микроархитектурата, използвана в повторенията на Pentium от Pentium Pro през Pentium III и Celeron и модулите на Pentium II Xeon и Pentium III Xeon .
* Intel пуска Nehalem(Core i7) през Ноември 2008, при който Hyper-Threading се завръща. Първото поколение Nehalem съдържа 4 ядра и ефикасно мащабира 8 нишки. От тогава се произвеждат два- и шест-ядрени модели, мащабирайки 4 и съответно 12 нишки.

Развитие на Hyper-Threading технологията

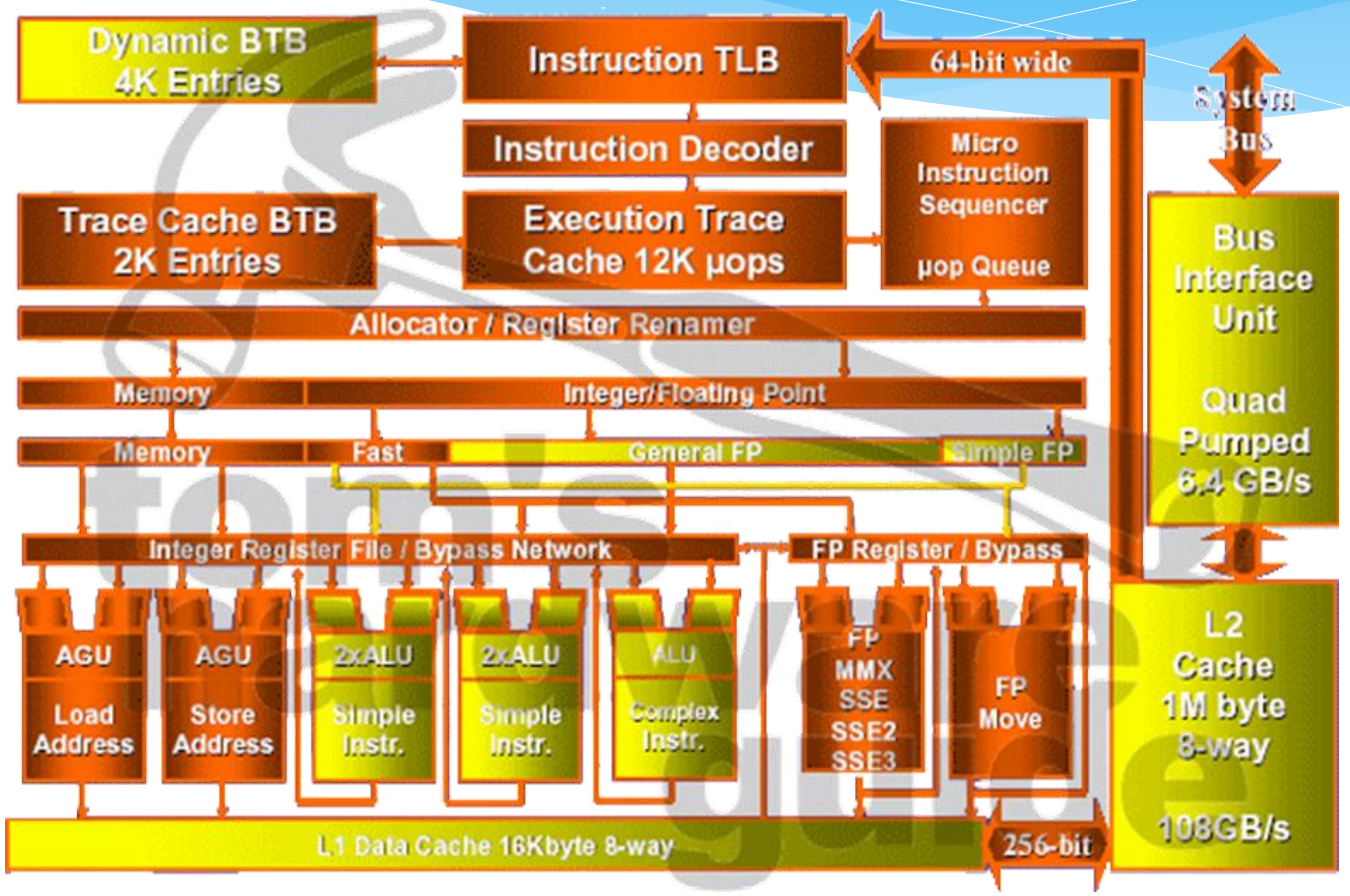
* Intel Atom процесор с Hyper-Threading за ниско консумиращи мобилни PCs и ниско-бюджетни десктоп PCs.
* Itanium 9300 стартира с 8 нишки за процесор(две нишки за ядро) чрез засилена Hyper-Threading технология. Poulson-следващо поколения Itanium е планиран да има допълнителни Hyper-Threading подобрения.
* Сървърският чип Intel Xeon 5500 също използва двупосочен Hyper-Threading.

Съвременни микроархитектури

NetBurst

Тази архитектура е наричана P68 в Intel-ските микроархитектури. Тази микроархитектура включва характеристики като Hyper Pipelined Technology и Rapid Execution Engine, които са първите в тази определена микроархитектура.

NetBurst


Dynamic BTB
4K Entries
Instruction TLB
64-bit wide
System Bus
Instruction Decoder
Micro Instruction Sequencer
µop Queue
Trace Cache BTB
2K Entries
Execution Trace Cache 12K µops
Bus Interface Unit
Quad Pumped 6.4 GB/s
Allocator / Register Renamer
Memory
Integer/Floating Point
Memory
Fast
General FP
Simple FP
Integer Register File / Bypass Network
FP Register / Bypass
AGU
Load Address
AGU
Store Address
2xALU
Simple Instr.
2xALU
Simple Instr.
ALU
Complex Instr.
FP MMX SSE SSE2 SSE3
FP Move
L2 Cache 1M byte 8-way
108GB/s
L1 Data Cache 16Kbyte 8-way
256-bit

NetBurst

Hyper Pipelined Technology(Свръх тръбопроводна технология)

Това име е дадено на 20 етапния инструкционен тръбопровод в Willamette ядрото. Това е значително увеличаване на броят на етапите в сравнение с Pentium III, който има само 10 етапа в неговия тръбопровод. Макар че по-дълбокият тръбопровод има увеличен брой непредсказани преходи, наказание за по-големият брой етапи в тръбопровода, позволява CPU да има по-висок тактова честота, която се е смятало за компенсация в загубите на производителността. По – малки инструкции за такт е непосредствено последствие от дълбочината на тръбопровода. Друг недостатък на наличието на повече етапи в тръбопровода е увеличение в броят на етапите, които трябва да бъдат проследени в събитието, където предсказателят на преходите прави грешка, увеличавайки наказанието, платено за непредсказване. За да отговори на този проблем, Intel създава Rapid Execution Engine и инвестира голяма част в неговата технология за предсказване на преходите, която, според Интел, намалява непредсказванията до 33% пред Pentium III.

NetBurst

Rapid Execution Engine

С тази технология, двата АЛУ в ядрото на CPU са двойно-увеличени, което означава, че те всъщност работят на два пъти на честотата на ядрото. Например при 3,8GHz процесор, АЛУ ще работи ефективно при 7,6 GHz. Причината за това е главно да се справят с ниският брой IPC(Instructions per Clock); в допълнение към това значително се подобрява цялостната производителност на CPU. Intel също high-speed barrel shifter с shift\rotate изпълнителна единица, която работи при същата честота като CPU ядрото. Недостатък е , че определени инструкции са сега много по-бавни от преди, затруднявайки оптимизацията за множество CPUs. Пример за това са сменящите и завъртащите операции, които страдат от липса на barrel shifter, които беше представен на всеки x86 CPU, започвайки с i386, също представен и в Athlon.

NetBurst

Execution Trace Cache

Вътре в L1 кешът на CPU, Intel включва Execution Trace Cache. Тя съхранява декодирани микрооперации, така когато изпълнява нова инструкция, вместо да извлича и декодира инструкцията отново, CPU директно достъпва декодираната микроинструкция от trace кеша, по този начин се спестява значително време. Освен това микрокод операциите са прихващани в техният предсказан път на изпълнение, което значи, че когато инструкциите са извлечени от CPU от кешът, те се представят в правилният ред на изпълнение.

NetBurst

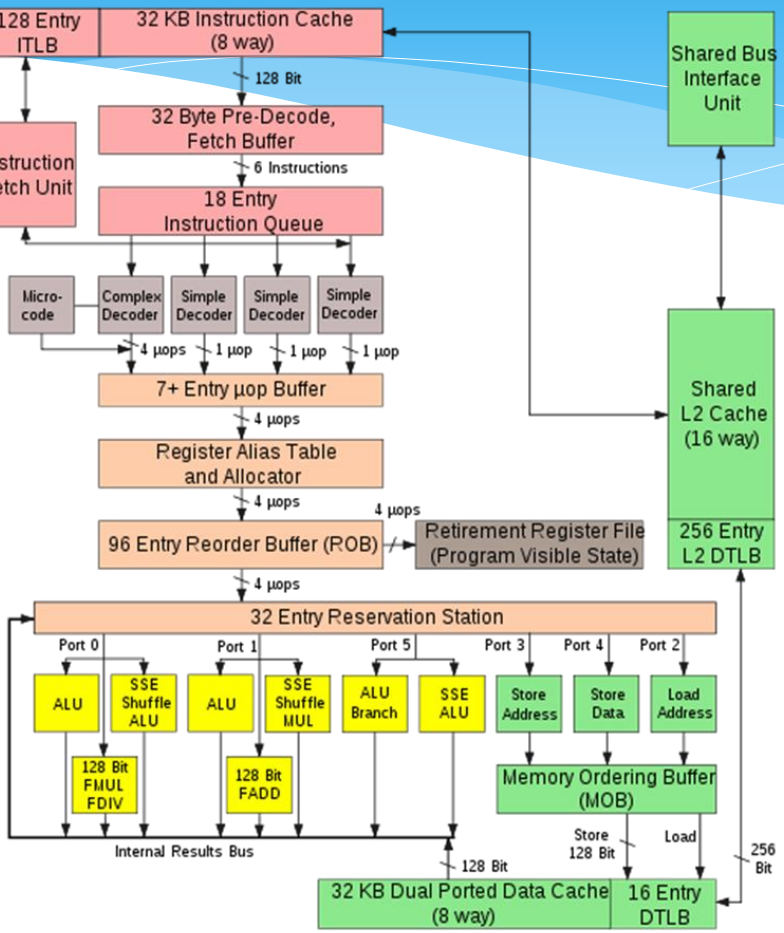
Въпреки тези подобрения, NetBurst архитектурата създава пречки за инженерите, опитвайки се да увеличат нейната производителност. С тази архитектура, Intel достигна тактова скорост от 10Ghz, но поради растящите тактови честоти, Intel се изправи пред пораждащи се проблеми, свързани с разсейваната мощност в приемливи граници. Intel достигна скоростна бариера от 3,8 Ghz през Ноември 2004, но са достигнали до проблеми, опитвайки се да достигнат дори това. Intel изоставя NetBurst през 2006 след като топлинни проблеми достигнали ниво на трудност и тогава развили **Core** микроархитектурата.

Съвременни микроархитектури

Core

Тази микроархитектура се връща до по-ниски тактови честоти и подобрява употребата на двата налични тактови цикъла и мощност в сравнение с предходната микроархитектура NetBurst. Core микроархитектурата осигурява по-ефикасни декодиращи етапи, изпълними единици, кешове и шини, намалявайки консумацията на мощност на Core 2 CPUs, докато увеличава техните производствени мощности. Процесорите на Intel променят значително консумацията на енергия според тактовата честота, архитектура и полупроводниковия процес.

Core



Intel Core 2 Architecture

Core

Core-базираните процесори не разполагат с Hyper-Threading микроархитектурата, намираща се в Pentium 4 процесорите. Това е така, защото Core микроархитектурата е потомък на P6 микроархитектурата, използвана от Pentium Pro, Pentium II, Pentium III и Pentium M.

Размерът на кеша L1 е уголемен от 32KB при Pentium II/III до 64KB L1 кеш/ядро(32KB L1 Данни + 32 KB L1 Инструкция) при Pentium M и Core/Core2. Липса L3 кеш, намиращ се в Gallatin core на Pentium 4 Extreme Edition, въпреки това L3 е налице при по-висок клас на Core-базирани Xeons. L3 кешът и Hyper-Threading са отново въведени в микроархитектурата Nehalem.

Core

Тази архитектура е 12-14 етапа дълга. Основната изпълнима единица е 4 броя, в сравнение с 3 при ядрата на P6, Pentium M и NetBurst микрокоархитектурите. Новата архитектура е с дву-ядрен дизайн със свързан L1 кеш и споделен L2 кеш, проектирани за максимална производителност за ват и подобрена мащабируемост.

Една нова технология, включена в дизайна, е Macro-Ops Fusion, която комбинира две x86 инструкции в единична микрооперация. Например, обща кодова последователност като сравнение, последвано от продължително прескачане, ще се превърне в единична микрокод операция.

Core

Други нови технологии включват 1 цикъл производителност(2 цикъла преди) на всичките 128 бита SSE инструкции и нов енергоспестяващ дизайн. Всички компоненти ще работят при минимална скорост, при необходимост, скоростта ще се покачва динамично. Това позволява на чипа да произвежда по-малко топлина и да консумира, колкото се може по-малко мощност.

Консумацията на мощност на тези нови процесори е изключително ниска-средната енергийна консумация е в границите на 1-2 вата при много ниските валотажни модели.

Core

Процесорни ядра

Процесорите при Core микроархитектурата могат да бъдат категоризирани по брой ядра, размер на кеша, сокет. Всяка комбинация от тези има уникално кодово име и продуктов код, които е използван сред редици марки.

Core

Conroe/Merom (65 nm)

Оригиналните Core 2 процесори са базирани около същите размери на чипа, които може да бъде идентифициран като CPUID(CPU IDentification) Family 6 Model 15. В зависимост от конфигурацията и пакетирането, техните кодови имена са Conroe (LGA 775, 4 MB L2 cache), Allendale (LGA 775, 2 MB L2 cache), Merom (Socket M, 4 MB L2 cache) и Kentsfield (Multi-chip module, LGA 775, 2x4MB L2 cache). Merom и Allende процесори с ограничени функции, могат да бъдат намери в Pentium Dual Core и Celeron processors, докато Conroe, Allendale и Kentsfield са продавани като Xeon процесори.

Допълнителни кодови имена за процесори базирани на този модел са Woodcrest (LGA 771, 4 MB L2 cache), Clovertown (MCM, LGA 771, 2x4MB L2 cache) и Tigerton (MCM, Socket 604, 2x4MB L2 cache, всички от които са на пазара само под марката Xeon.

Core

Penryn/Wolfdale (45 nm)

Интел през 2007/2008 свива архитектурата на Core до 45 nm като CPUID модел 23. При Code 2 процесори, той се използва с кодови имена Penryn (Socket P), Wolfdale (LGA 775) и Yorkfield (MCM, LGA 775), някой от които също са продавани като Celeron, Pentium и Xeon процесори.В марката Xeon, кодовите имена Wolfdale-DP и Harpertown се използват за LGA 771 базирани MCMs, с две или четири активни ядра Wolfdale.

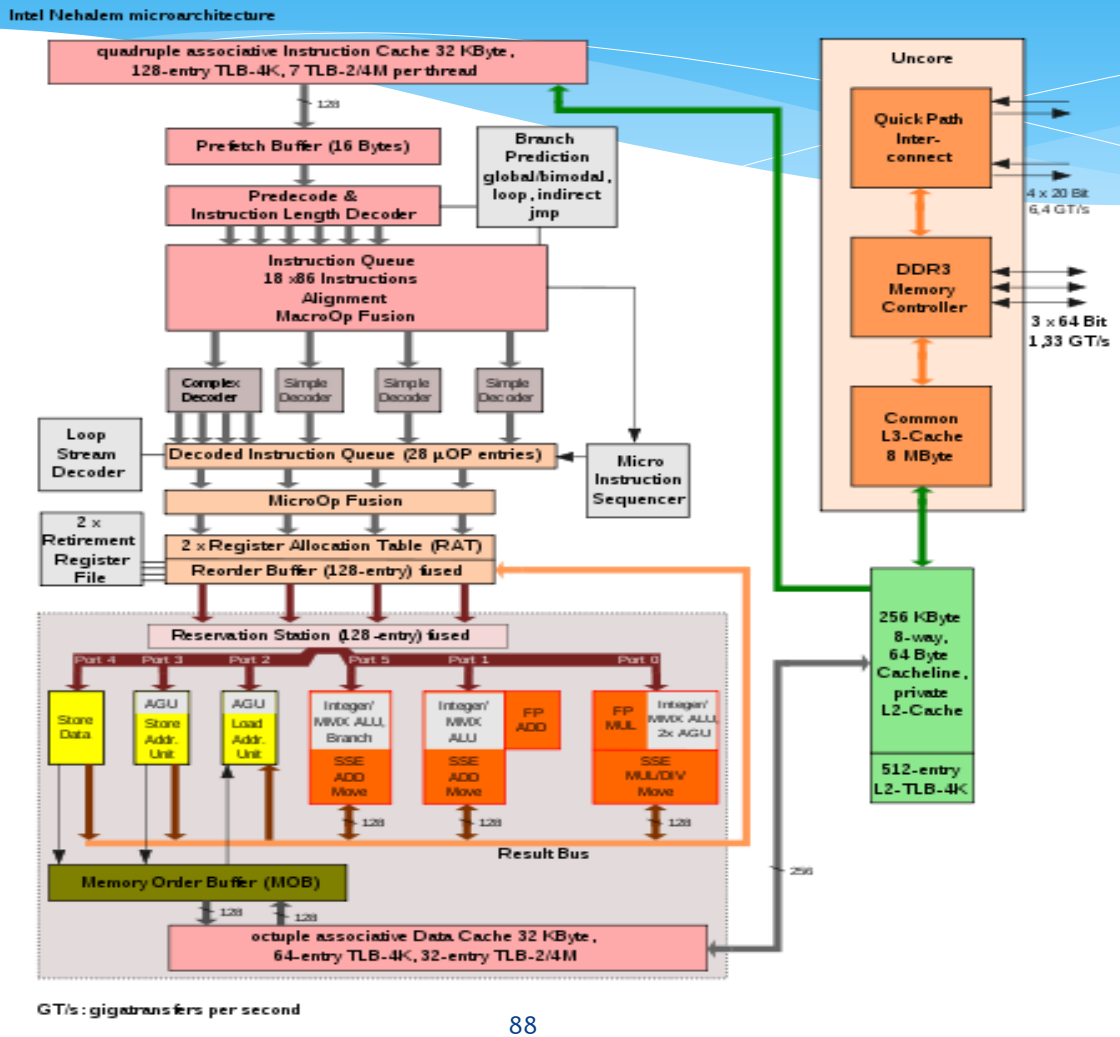
Чиповете са в два размера, с 6MB и 3 MB L2 кеш. По-малката версия е често наричана Penryn-3M и Wolfdale-3M, както и съответно Yorkfield-6M. Едно-ядрената версия на Penryn, не е отделен модел като Merom-L, но е версия на модела Penryn-3M само с едно активно ядро.

Съвременни микроархитектури

Nehalem

Nehalem е кодово име за Интелската процесорна микроархитектура, наследник на Core микроархитектурата, използва 45nm процес. Първият процесор с Nehalem архитектура е десктопският Core i7, който бе представен през Ноември 2008. Nehalem се отнася до напълно различна архитектура от NetBurst, макар че имат някой общи неща. Nehalem- базираните микропроцесори използват по-високи тактови честоти и са по-енергийно ефективни от Penryn микропроцесорите. Hyper-Threading е въведен отново с намаляване на L2 кеша, който е включен като L3 кеш, който е използван от всички ядра. Nehalem е заместен със Sandy Bridge микроархитектурата през Януари, 2011.

Nehalem


Intel Nehalem microarchitecture
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU
Store Addr. Unit
AGU
Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octuple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K
GT/s: gigatransfers per second

Nehalem

* ***Технология***
* Въведен отново Hyper-Threading
* 4-12 MB L3 кеш
* Второ ниво предвиждане на преходите и translation lookaside buffer
* Всички процесорни ядра са върху един чи, quad и octo-core процесори
* 64 KB L1 кеш/ядро(32 KB L1 Данни + 32 KB L1 Инструкция) и 256 KB L2 кеш/ядро
* ***Подобрения на производителността и мощността***
* Nehalem се фокусира върху производителността, това води до увеличения размер на ядрото. Спрямо Penryn, Nehalem има :
* 10-25% повече еднонишково изпълнение/ 20-100% повече многонишково изпълнение на същото ниво на мощност
* 30% по-ниско потребления на енергия за същата производителност
* 15-20% такт-за-такт увеличаване на производителността за ядро

Nehalem

Westmere

Това е името, дадено на 32 nm размер на Nehalem. Тези процесори са стартирани през Януари, 2010.

Подобрени черти при Westmere, спрямо Nehalem:

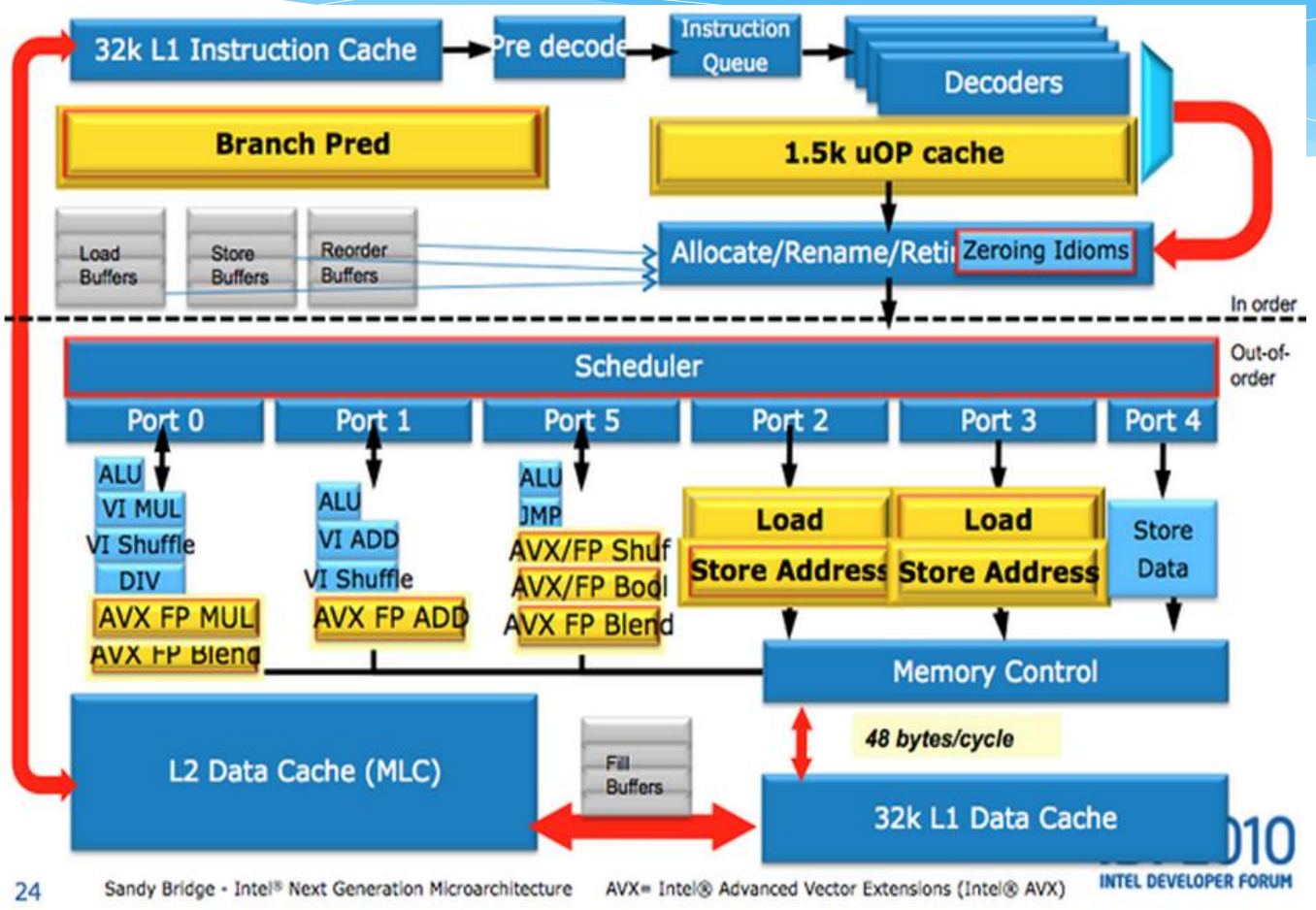
* Прости 6-ядрени и 10-ядрени процесори
* Нов набор инструкции, които дават над 3 пъти по-добре крептирани и разкрептирани (AES – Advances Encryption Standard) процеси, сравнено с преди
  * Доставя 7 нови инструкции, които ще бъдат използвани от AES алгоритъма. Тези инструкции ще позволят на процесора да изпълни хардуерно-ускорено крептиране
* Вградени графики, добавено в пакета на процесора
* Подобрена латентност на виртуализацията
* Нова виртуализационна съвместимост, която позволява 16-bits гости да изпълняват
* Поддръжка за “Huge pages” от 1 GB

Съвременни микроархитектури

Sandy Bridge

* Това е кодово име за микроархитектура, разработена от Intel, даваща начало през 2005 за CPU в компютрите да замени Nehalem микроархитектурата. Intel демонстрира Sandy Bridge процесор през 2009 и изкарва на пазара първите продукти, базирани на тази технология през Януари, 2011 под марката Core.
* Производственият процес на 32 nm е базиран на planar double-gate transistors.
* Тази технология използва равнинни производствени процеси да създаде double-gate устройства, избягвайки по-сурови литографски изисквания, свързани с не-равнинните, вертикални транзисторни структури. При равнинните double-gate транзистори, drain-source канал е затворен между два независимо изфабрикувани gate/gate оксидни стакове. Главното предизвикателство при правенето на такива структури е постигане на задоволително съответствие между горните и долните порти.

Sandy Bridge



Sandy Bridge

Подобрения при Sandy Bridge спрямо Nehalem:

- 32 KB данни + 32 KB инструкция L1 кеш(3 такта) и 256 KB L2 кеш(8 такта) за ядро
- Споделен L3 кеш
- 64-битова кеш линия
- Две load/store операции за CPU цикъл за всеки канал на паметта
- Декодиран кеш на микрооперации и уголемен, оптимизиран предсказател на преходите
- Възможност до 8 физични и 16 логични ядра чрез Hyper-Threading
- Подобрена производителност за транседална математика, AES крептиране
- Допълнителни векторни разширения(AVX) 256bit инструкционен набор с по-широки вектори, нов разтегателен синтаксис и богата функционалност
- Средната производителност от такт до такт се е покачила 11,3% в сравнение с Nehalem.

Развитие в бъдеще:

Ivy Bridge микроархитектура :

- 22nm, базиран на 3-gate транзистори
- Подобрения спрямо Sandy Bridge:
  - 6-16 изпълними единици (Sandy Bridge 6-12)
  - До 50% по-малко енергийна консумация на същото ниво на производителност
  - От 5 до 15% увеличение на производителността на CPU
  - От 20 до 50% увеличение във вградената GPU производителност
- Max. CPU clock rate : 2.5 GHz to 3.5 GHz
- Стартиране на производство 29 Април 2012

Развитие в бъдеще:

Haswell микроархитектура:

- Наследник на Ivy Bridge
- 22nm производствен процес
- 3D 3-gate транзистори
- 14-етапен тръбопровод
- 128 bytes кешова линия
- Нова допълнителна енергийно-спестяваща система
- 32nm Platform Controller Hub

Развитие в бъдеще:

Skylake микроархитектура

- Наследник на Haswell
- 14nm
- Очакван през 2015/2016

Извод

С напредване на технологичният прогрес, тенденциите при микроархитектурните дизайни са :

- Намаляване на размера на чиповете
- Увеличаване на фазите в тръбопровода
- Увеличаване на изпълнимите единици
- Намаляване на консумацията на енергия
- Увеличаване на производителността на CPU и GPU
- Увеличаване на размера на кешовете и кешовата линия
- Увеличаване на скоростта на обмен на информация с периферните устройства и главната памет