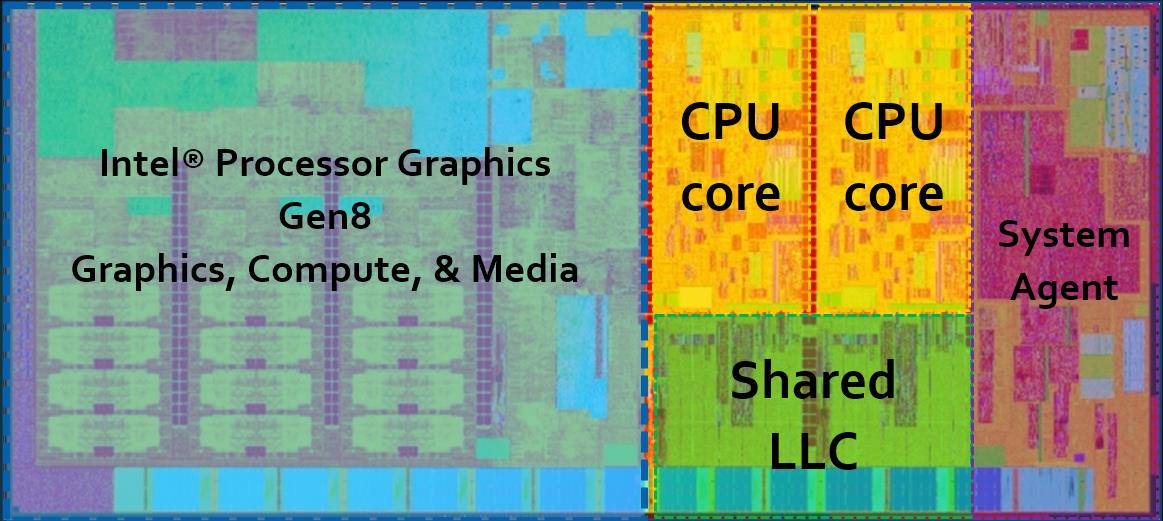
1. **Sprzętowe architektury heterogeniczne**
   1. **Architektura CPU-GPU Intel CoreTM, AtomTM**

Są to dwie rodziny wielordzeniowych procesorów, w których zastosowano system SoC [1], rozwiązanie to wykorzystuje urządzenia takie jak procesory CPU oraz procesory GPU, do wykonywania obliczeń heterogenicznych oraz przetwarzania mediów. Schemat układu Intela jest pokazany poniżej.



Powyższy schemat pokazuje, że architektura łączy wiele procesorów CPU Intela, procesor graficzny Intela oraz inne stałe funkcje na jednolitej krzemowej matrycy. Architektura implementuje unikatowe zakresy czasowe, zegar rdzenia CPU, zegar procesora graficznego oraz pierścieniowo połączony zegar. Architektura SoC została zaprojektowana z możliwością jej rozszerzania. Druga generacja Intel@ CoreTM wprowadził możliwość heterogenicznego przetwarzania mediów wprowadzając Intel® Quick Sync Video,  Intel® Media SDK. W trzeciej generacji procesorów Intel@ CoreTM kontynuuje wsparcie dla heterogenicznego przetwarzania mediów oraz zapewnia wsparcie dla OpenCL 1.2. Natomiast nowe procesory Intel@ Core M wprowadziły ułatwienia w programowaniu w OpenCL 2.0, architektura również wspiera wspólną fizyczną pamięć a co za tym idzie wspólne dane aplikacyjne pomiędzy procesorem CPU a procesorem graficznym. Wspólna obsługa pamięci wirtualnej OpenCL 2.0 pozwala aplikacjom na dzielenie struktur danych płynnie pomiędzy dwoma urządzeniami. Architektura ta zapewnia środowisko dla aplikacji, poprzez dzielenie zasobów między procesory CPU a procesory graficzne.



Rysunek 1 Model głównego algorytmu

Niech Ti będzie czasem wykonania i-tego kroku, wtedy można oszacować całkowity czas wykonania algorytmu równy: ∑Ti. Proste sposoby optymalizacji to dostrojenie wydajności w każdym kroku indywidualnie poprzez dostosowanie kodu skalarnego, wielowątkowości, wektoryzacji (Intel posiada specjalne narzędzie pozwalające na odpowiednią wektoryzację) lub odciążyć procesor graficzny. To podejście może być pierwszym krokiem optymalizacji jednak może spowodować ograniczenie możliwości optymalizacji wydajności w dalszych krokach. Dla przykładu optymalizując jeden z pięciu kroków, sprawimy że będzie on dwa razy szybszy jednak jeżeli chodzi o całość to może to przynieść wzrost wydajności jedynie o 10% w stosunku do całości jeśli wszystkie etapy wykonują się w tym samym czasie.

* 1. **FPGA - Field-Programmable Gate Array**

Jest to układ scalony osadzony na płytce krzemowej, który zawiera tablicę konfigurowalnych bloków logicznych. FPGA może być przeprogramowana w taki sposób, że może wykonać różne funkcje w przeciągu kilku mikrosekund. Przed zaprogramowaniem FPGA nie wie w jaki sposób komunikować się z otaczającymi ją urządzeniami, jest to znaczne zwiększenie elastyczności programowania a co za tym idzie także zwiększenie jego złożoności. Sprzedawcy sprzętu komputerowego dostrzegli potencjał tego układu i zaczęli włączać go do oferowanych przez siebie systemów HPC, gdzie mogą one działać jako co-procesory przyspieszające kluczowe jądra wewnątrz aplikacji. Wiele firm zastosowało FPGA do stworzenia własnych programowalnych systemów, jedną z takich firm była Cray, włączyła ona FPGA do systemu HPC i nazwała ten układ XD1. Na konkurenta Cray wyrosła firma SGI, która wprowadziła na rynek system RASC Module oraz później RASC RC100 Blade. Firma SRC computers wyprodukowała własne rekonfigurowalne systemy z procesorami typu MAP. Wszystkie te systemy bazują na FPGA, które są dostarczane przez firmę Xilinx, jest do największy producent FPGA na świecie. Firma Xilinx wprowadziła serię Virtex jest to seria FPGA zaadoptowana do systemów HPC. Wzrost zainteresowania FPGA nałożył się na okres walki mikroprocesorów aby nadążyć za prawem Moore’a. Zaobserwowano spadek wzrostu wydajności w tworzonych nowych mikroprocesorach podczas gdy koszt optymalizacji ich zapotrzebowania na moc cały czas rośnie tak samo sprawiło, że firmy zaczynają poszukiwać innych rozwiązań technologicznych, które bardziej by im się opłacały. Użycie FPGA w systemach HPC może przynieść trzy wyraźne korzyści w porównaniu do konwencjonalnych klastrów komputerowych. Pierwsza zaleta jest taka, że FPGA pobiera mniej energii niż klaster, druga użycie FPGA jako akceleratora może znacznie zwiększyć gęstość obliczeń, trzecia FPGA może zapewnić znaczny wzrost wydajności dla pewnego zbioru aplikacji.

* 1. **XD1**

XD1 jest to system stworzony przez firmę Cray we współpracy z Oak Ridge National Laboratory. Układ XD1 zawiera dwanaście płyt, na każdej z nich znajduje się dwanaście procesorów AMD Operon 248, które pracują z częstotliwością 2,2 GHz oraz każdemu procesorowi przysługuje 4 GB pamięci. Z tego wynika, że system zawiera 144 procesory, 576 GB pamięci oraz 18 TB na dysku. Układ XD1 połączył ze sobą nowe rozwiązania dotyczące łączności, narzędzie zarządzające oraz rekonfiguracyjne technologie obliczeń z procesorem AMD Operon.

1. **Zjawisko kradzieży pracy w obliczeniach równoległych**
   1. **Idea**

Algorytmy zarządzania wątkami dążą do tego aby pewna liczba wątków była działająca co prowadzi do tego, że procesory są zajęte, zadaniem algorytmów jest optymalizacja ilości wątków aby ich liczba nie była zbyt duża gdyż duża ilość wątków zajmuje dużą ilość pamięci co może znacznie wydłużyć czas obliczeń. Istnieją dwa sposoby aby pogodzić te wymagania pierwszy sposób to podział pracy a drugi to kradzież pracy.

Podział pracy - idea jest następująca: algorytm próbuje przenieść kilka wątków z jednego procesora na drugi w celu rozłożenia pracy, jednak gdy przeniesiony wątek jest powiązany z innymi wątkami musi on dodatkowo informować inne wątki o postępach swojej pracy co jest problematyczne.

Kradzież pracy – w przypadku kradzieży pracy niewykorzystywane procesory przejmują inicjatywę i próbują wykraść wątki procesorom zajętym, W algorytmach tego typu migracja wątków występuje rzadziej niż w przypadku podziału pracy ponieważ gdy zajęte są wszystkie procesory to migracja nie występuje, natomiast w przypadku podziału pracy migracja ta występuje zawsze. Początki idei kradzieży pracy sięgają czasów gdy Burton i Sleep pracowali nad znalezieniem równoległego rozwiązania wykonywania programów funkcjonalnych. Ci programiści pokazali jakie korzyści nie się za sobą kradzież pracy w odniesieniu do komunikacji i przestrzeni. Rudolph, Slivkin-Allalouf analizowali algorytm losowej kradzieży pracy dla rozwiązania problemu niezależnego rozłożenia obciążenia pracy na komputerach równoległych. Karp i Zang pracowali nad algorytmem losowej kradzieży pracy, który miał być zastosowany do problemu wstecznego wyszukiwania równoległego. Zhang wraz z Ortynskim otrzymali akceptowalny wynik dla wymagań komunikacyjnych tego algorytmu. Wszyscy wyżej wymienieni naukowcy przyczynili się do rozwoju algorytmu kradzieży pracy.

* 1. **Przegląd algorytmów**
     1. **Teoria zachłannego planowania**

Gdzie:

X – plan wykonywany na procesorze P

T(X) – czas wykonania planu

- suma wszystkich instrukcji wykonywanych przez wątki

- jest to suma wszystkich zaalokowanych pamięci przez przodków wątku wraz z jego pamięcią

- jest to długość ścieżki krytycznej czyli najdłuższej ścieżki w skierowanym grafie acyklicznym

* + 1. **Algorytm zajętych liści**

Treść algorytmu jest następująca: dla P procesorów, które wykonują obliczenia wielowątkowe o pracy T1 z krytyczną długością ścieżki T∞ oraz gdzie głębokość stosu wynosi S1 plan wykonania X realizuje czas oraz jednocześnie. Algorytm działa następująco: przed krokiem tth algorytm obliczył i wykonał t – 1 kroków planu wykonywania. W kroku tth algorytm aby obliczyć ten krok używa tylko informacji z porcji obliczeń, które jak dotąd zostały wykonane, w szczególności nie używa żadnych informacji pochodzących z jeszcze nie wykonanych instrukcji. Algorytm trzyma wszystkie żyjące wątki w globalnej puli, która jest dostępna dla wszystkich procesorów. Gdy dojdzie do utworzenia wątku dziecka, zostaje ono dodane do globalnej puli. Algorytm w danym kroku tth pozwala procesorom na dodawanie wątków do puli oraz usuwanie ich z niej, rywalizacja procesorów o dostęp do puli jest ignorowana.

Algorytm zaczyna się od głównego wątku w globalnej puli oraz od stanu bezczynności wszystkich procesorów. Na początku każdego kroku każdy procesor albo jest bezczynny albo posiada wątek, który wykonuje, bezczynne procesory starają się zdobyć dostęp do globalnej puli wątków i usunąć z niej wątek aby rozpocząć jego wykonywanie. Jeżeli w puli jest wystarczająco dużo wątków ile jest procesorów wtedy każdy procesor pobiera jeden wątek z puli i zaczyna wykonywanie go, w innym przypadku procesory, które nie pobrały z puli żadnych wątków pozostają w stanie bezczynności. Jeżeli procesor posiada wątek wtedy w każdym kroku wykonuje jedną instrukcję tego wątku aż do momentu aż wątek nie utworzy potomka, zablokuje się lub zginie. W przypadku tych trzech zachowań procesor zachowuje się następująco:

Tworzenie nowego potomka – procesor kończy bieżący krok i zwraca wątek rodzica do globalnej puli wątków, natomiast w następnym kroku pracuje już na wątku potomku.

Zablokowanie – procesor kończy bieżący krok zwracając wątek zablokowany do globalnej puli, następny krok procesor rozpoczyna w stanie bezczynności.

Wątek ginie – jeśli wątek zginął procesor kończy bieżący krok sprawdzając czy rodzic wątku posiada jakichś żywych potomków. Jeżeli wątek rodzic nie posiada żywych potomków i jeśli żadne inny procesor nie pracuje na tym wątku wtedy procesor pobiera wątek z puli i w następnym kroku rozpoczyna wykonywanie instrukcji tego wątku. W przeciwnym wypadku, w następnym kroku procesor przechodzi w stan bezczynności.

* + 1. **Algorytm losowej kradzieży pracy**

W tym algorytmie każdy procesor zawiera własną kolejkę wątków, które są po kolei wykonywane. Kolejka procesora ma dwa końce górę i dół, wątki są dodawane na dół kolejki i także z niego usuwane. Kolejka procesorów przyjmuje postać stosu wywołań dodając i usuwając wątki z dołu. Natomiast wątki, które migrują do innego procesora są usuwane z góry. Algorytm działa następująco: procesor pobiera ze swojej kolejki wątek i rozpoczyna wykonywanie instrukcji znajdujących się w nim, kontynuuje wykonywanie tych instrukcji do czasy gdy wątek nie utworzy wątku potomnego, zablokuje się, zginie lub pozwoli na zablokowanie wątku.

Wątek tworzy potomka – procesor umieszcza wątek rodzica na dole kolejki i rozpoczyna wykonywanie wątku potomka.

Wątek blokuje się – jeśli wątek się zablokuje procesor sprawdza kolejkę, jeśli zawiera ona jakieś wątki wtedy procesor pobiera wątek z samego dołu i rozpoczyna wykonywanie go. Jeśli kolejka jest pusta procesor rozpoczyna proces kradzieży pracy czyli pobiera wątek na samej górze losowo wybranego procesora i rozpoczyna wykonywanie go.

Wątek ginie – jeśli wątek zginie procesor wykonuje regułę drugą.

Pozwolenie na blokadę wątku – jeśli wątek pozwoli na blokadę wątku potomka wątek potomek jest przeniesiony na sam dół kolejki procesora, który pracuje na wątku rodzicu.

Wątek może jednocześnie umożliwić blokadę innego wątku i samemu się zablokować lub zginąć, w tym wypadku wykonuje najpierw regułę cztery umożliwiając blokadę innego wątku a następnie regułę dwa gdy sam zostanie zablokowany albo regułę trzy gdy zginie. Algorytm zaczyna się ze wszystkimi kolejkami pustymi. Wątek główny jest umieszczany w kolejce jednego procesora podczas gdy inne procesory rozpoczynają proces kradzieży pracy. Procesor rozpoczyna proces kradzieży pracy w następujący sposób: procesor staje się złodziejem i próbuje wykraść wątek od procesora ofiary, który został wybrany jednolicie i losowo. Złodziej wysyła zapytanie do kolejki ofiary i jeżeli kolejka nie jest pusta usuwa wątek znajdujący się na samej górze kolejki. Jeśli kolejka jest pusta złodziej próbuje znowu pobrać wątek od kolejnego losowo wybranego procesora ofiary.

* + 1. **Recycling game**

Jest to kombinatoryczna gra, którą rozgrywają między sobą przeciwnicy. W grze mamy piłki, które są wrzucane przez przeciwników do losowy wybranego kosza. Parametr P jest to ilość piłek w grze, liczba ta jest równa liczbie koszy biorących udział w grze. Liczba M jest to całkowita liczba piłek wrzuconych, przez przeciwników. Na początku wszystkie piłki znajdują się w zbiorniku, są oddzielone od koszy. W każdym kroku działania algorytmu przeciwnik wykonuje sekwencyjnie następujące dwie operacje:

Przeciwnik pobiera ze zbiornika pewną liczbę piłek usuwając je ze zbiornika, może pobrać wszystkie lub może nie pobrać żadnej. Następnie dla każdej pobranej piłki losowo wybiera kosz z puli P koszy i wrzuca do niego piłkę.

Przeciwnik sprawdza każdy kosz po kolei i jeżeli kosz zawiera przynajmniej jedną piłkę usuwa wszystkie piłki z kosza i zwraca do zbiornika.

Jest dozwolone że Przeciwnik może wrzucić M liczbę piłek do kosza. Algorytm się kończy gdy zostanie wrzuconych M piłek oraz wszystkie piłki zostaną usunięte z koszy i umieszczone w zbiorniku. Algorytm ten modeluje doglądanie żądania kradzieży w algorytmie kradzieży pracy. Można to zobaczyć przyjmując, że piłki i kosze są własnością odrębnego procesora, jeśli piłka znajduje się w zbiorniku, znaczy to że właściciel piłki nie tworzy żądania kradzieży. Jeżeli piłka znajduje się w koszu oznacza to, że właściciel piłki stworzył żądanie kradzieży i umieścił je na liście właściciela kosza, ale żądanie nie zostało jeszcze wykonane. Gdy piłka zostanie usunięta z kosza i zwrócona do zbiornika oznacza to, że żądanie kradzieży zostało wykonane.