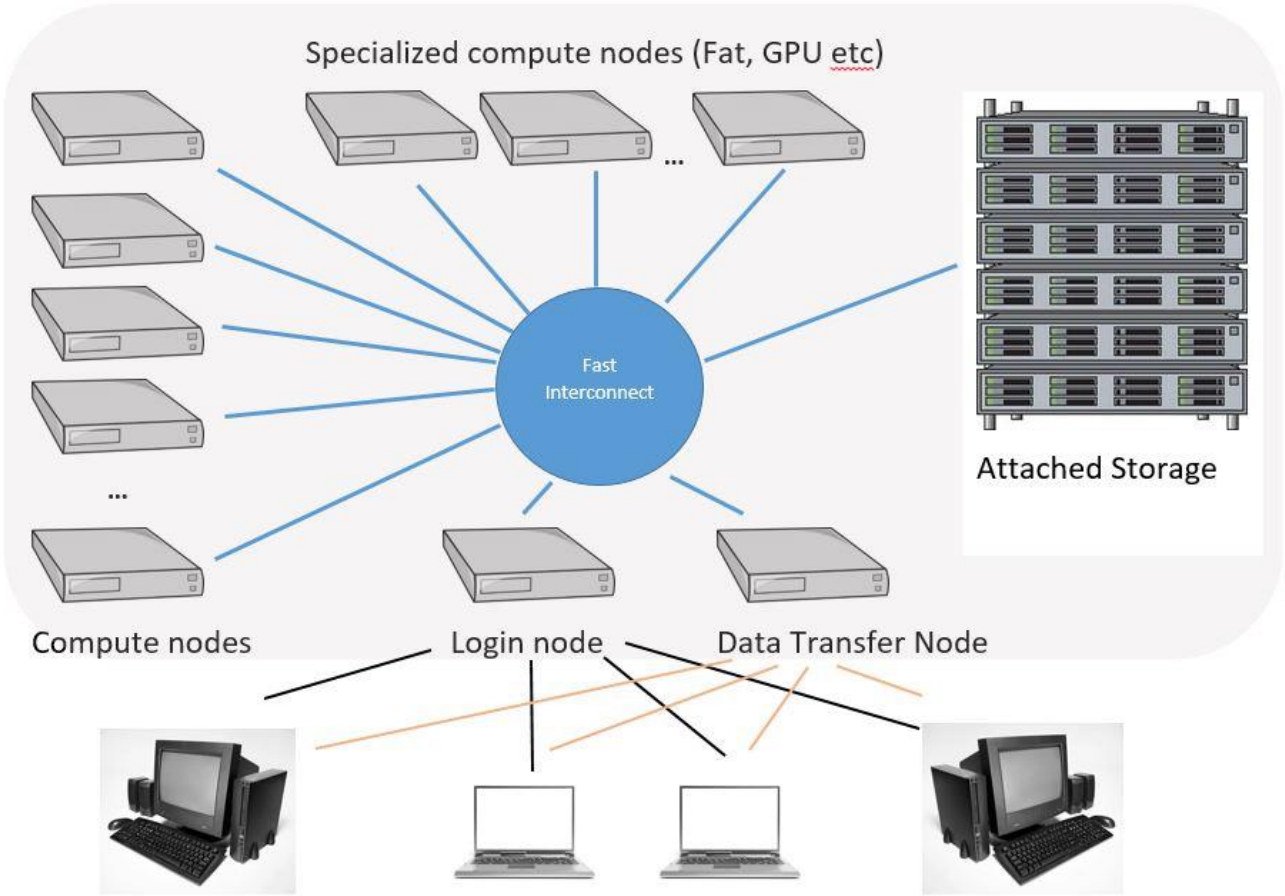
Programowanie równoległe

Wstęp do programowania równoległego

Informacje wstępne

- Koncepcja równoległości - po co nam to?
- Jedno urządzenie - wiele zadań
- Wiele zadań - jeden nadzorca (procesor)
- Idea programowania urządzeń wykonujących kawałek większych obliczeń (klaster obliczeniowy)
- Idea wieloprocesrowości w układach elektrocznicznych
- Idea wielordzeniowości w układach elektronicznych
- Idea wielozadaniowości (potokowość) w układach elektronicznych

Klaster obliczeniowy



https://www.hpc.iastate.edu/sites/default/files/uploads/HPCHowTo/HPCCluster.JPG

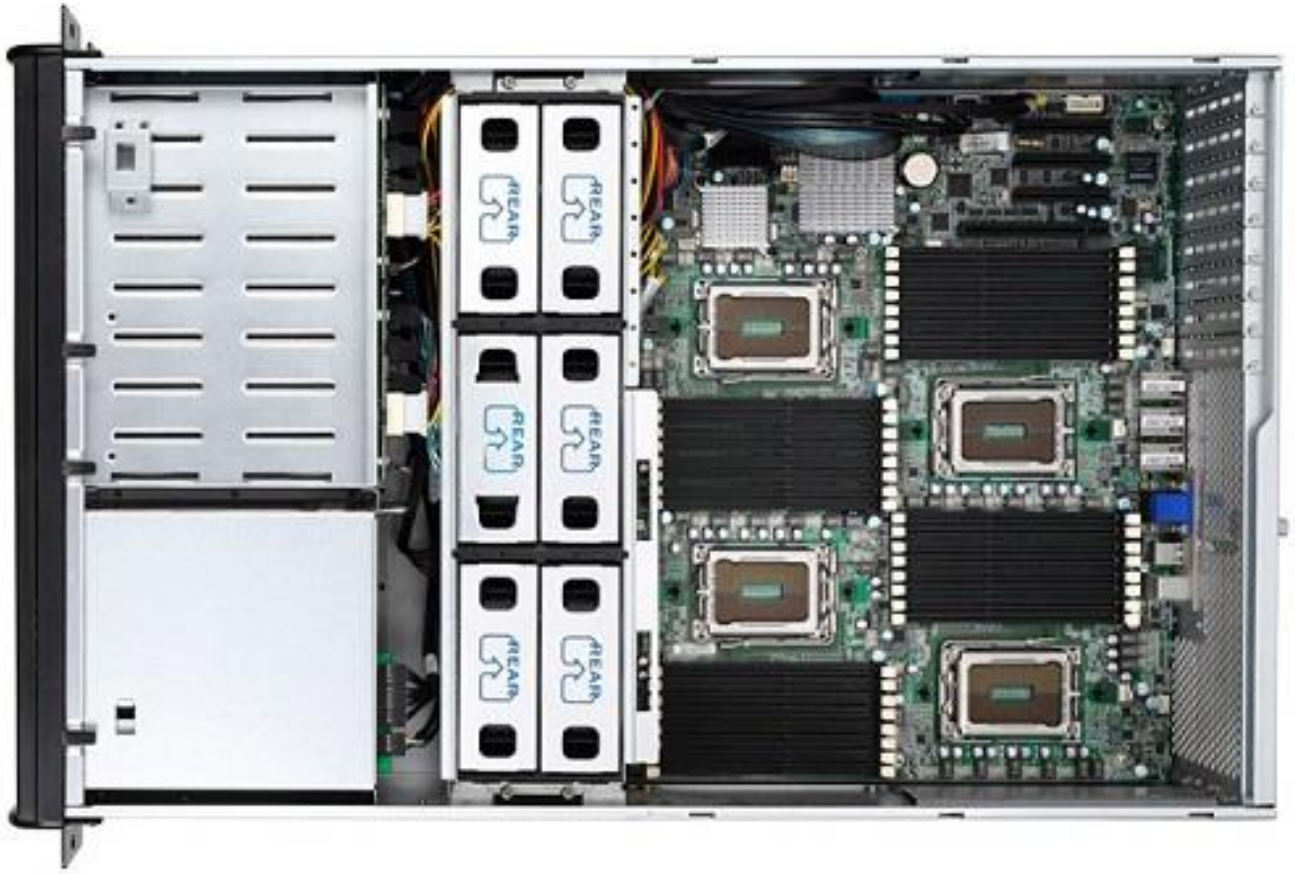
Klaster obliczeniowy

https://www.climatemodeling.org/~forrest/osdj-2000-11/image01.jpg

Klaster obliczeniowy

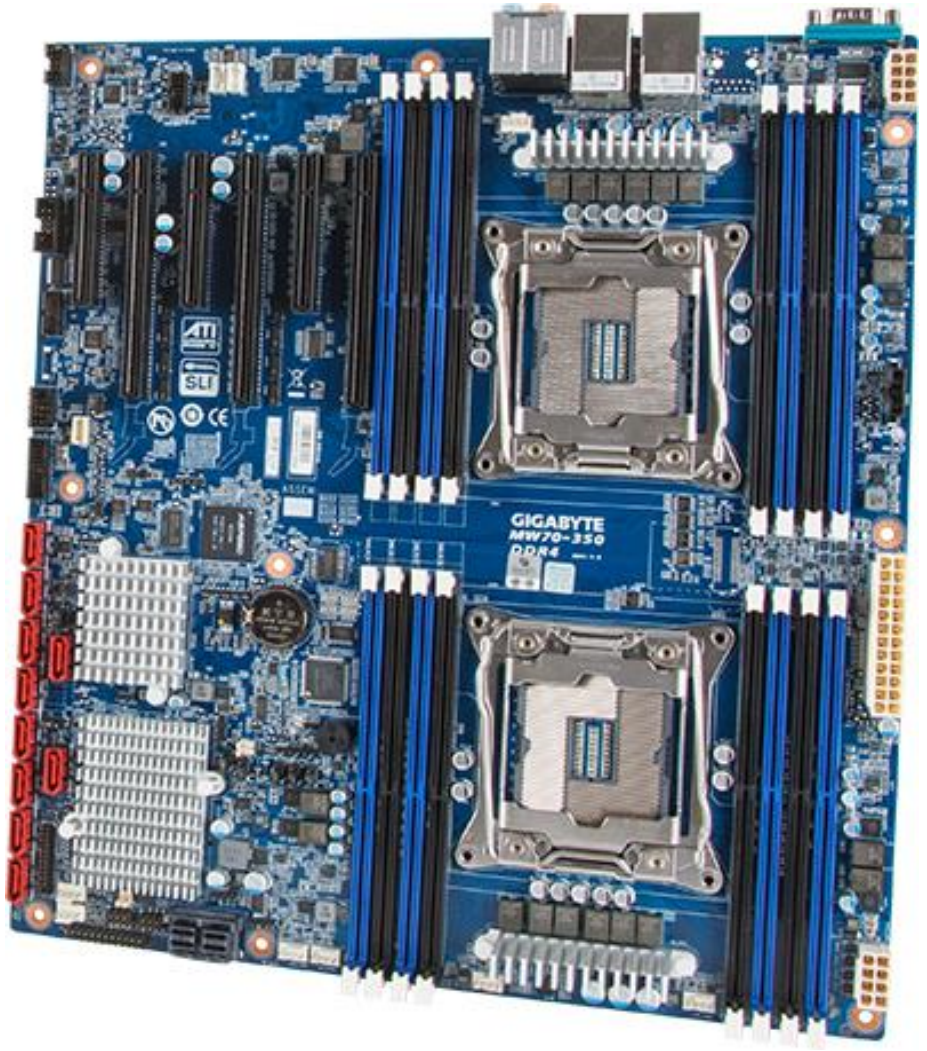
https://ncan.us/wp-content/uploads/2019/07/original.jpg

Wieloprocesorowość



https://www.custom-build-computers.com/image-files/using-server-motherboard-for-gaming.jpg

Wieloprocesorowość

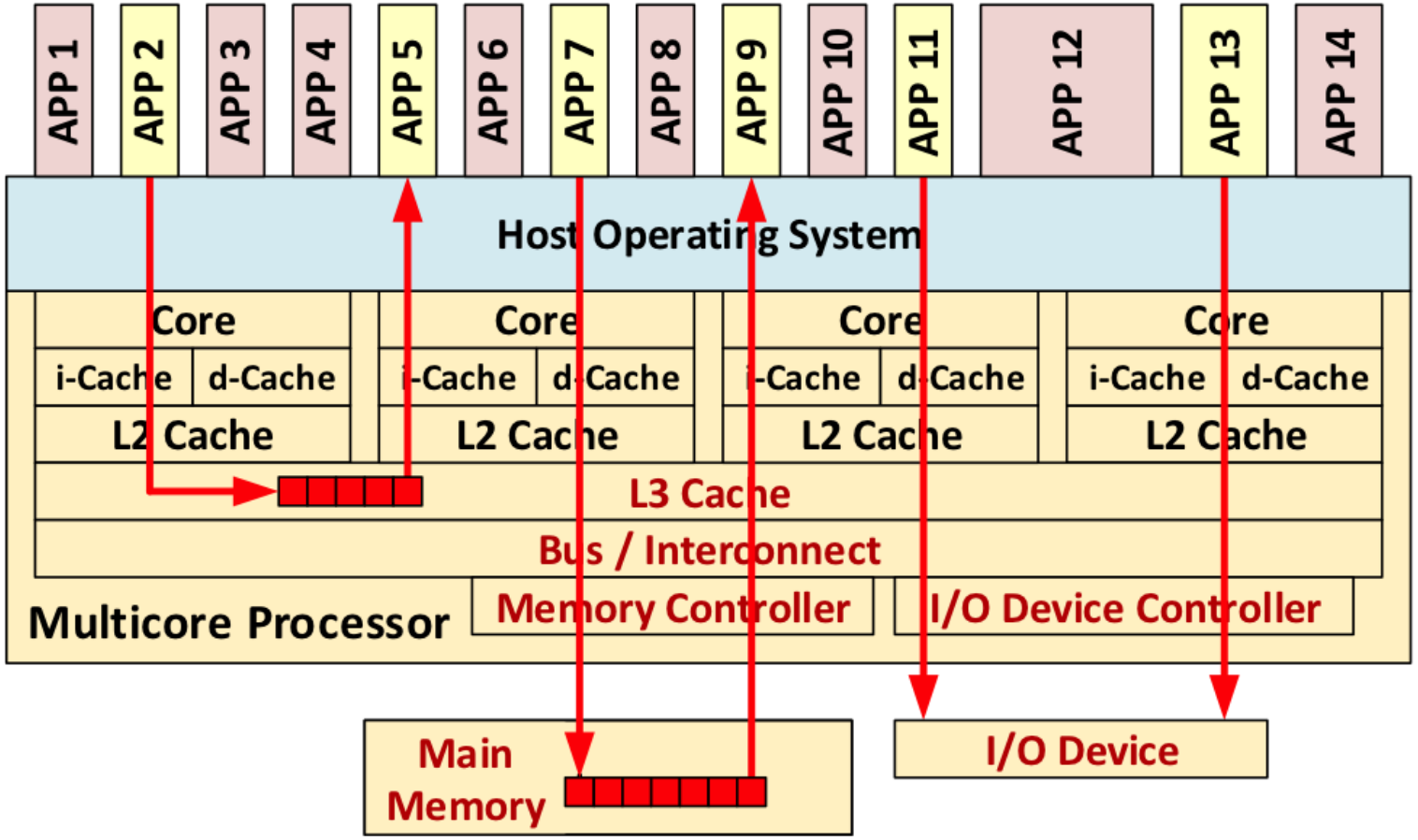


https://www.gigabyte.com/FileUpload/Global/news/1367/MW70-3S0-Angle.jpg

Wieloprocesorowość

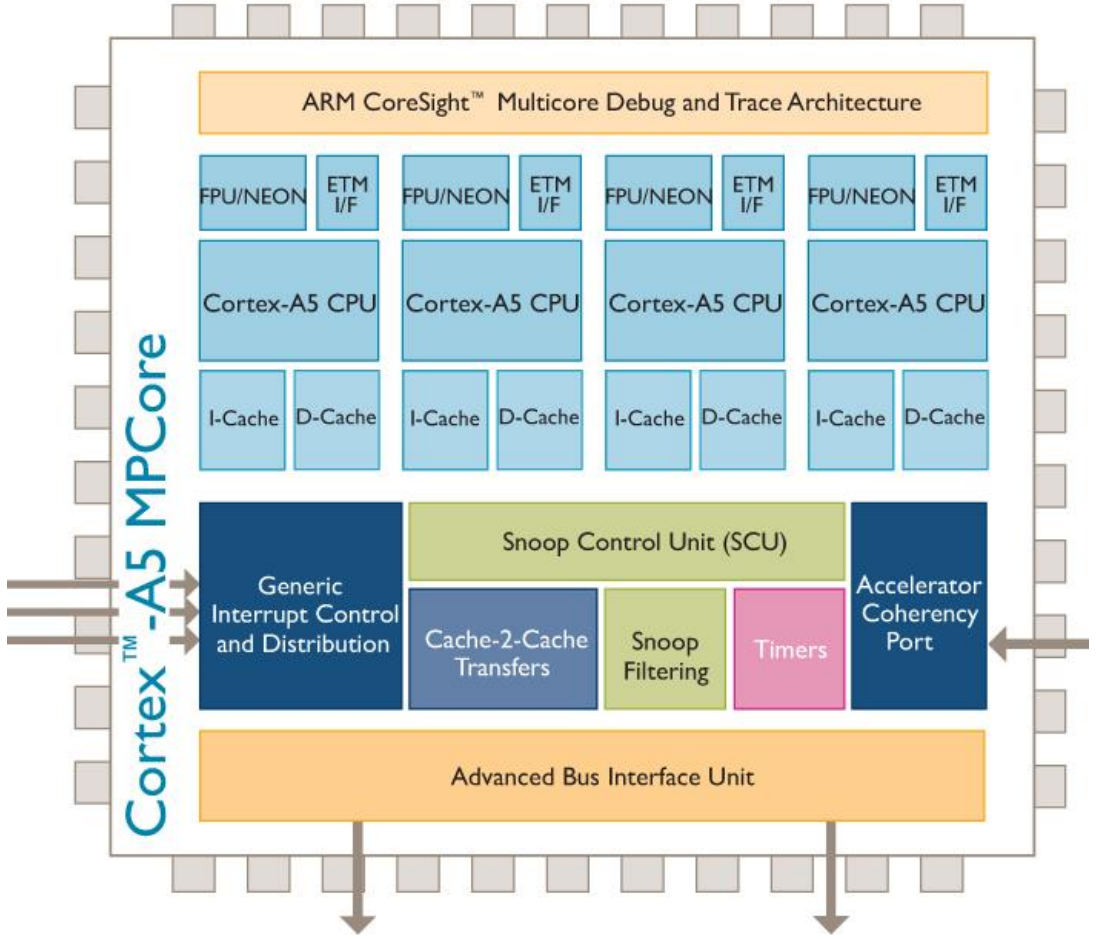
https://images-na.ssl-images-amazon.com/images/I/71ltDKbW89L._AC_SX522_.jpg

Wielordzeniowość



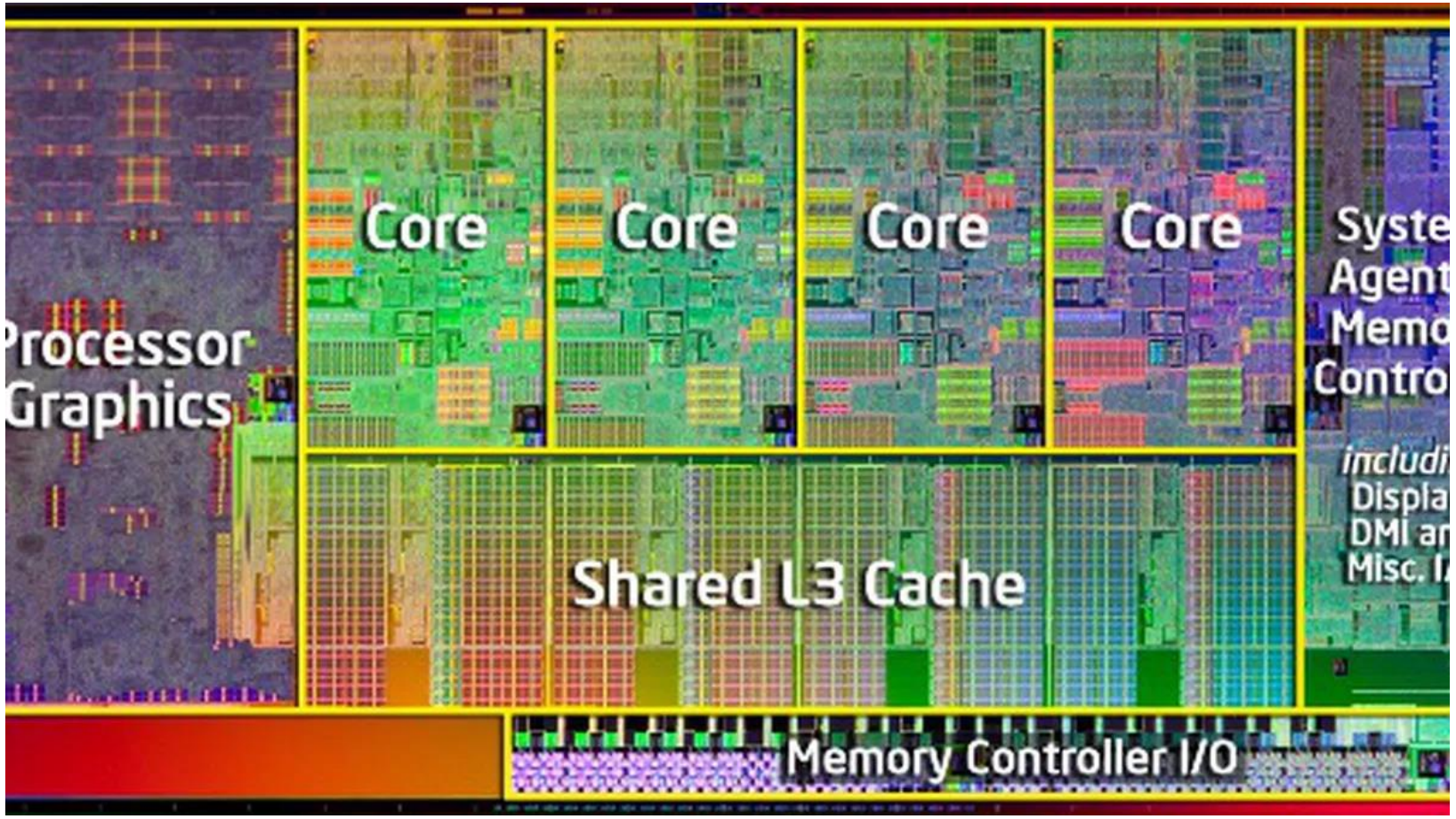
https://insights.sei.cmu.edu/sei_blog/firesmith_secondmulticore_figure3.png

Wielordzeniowość



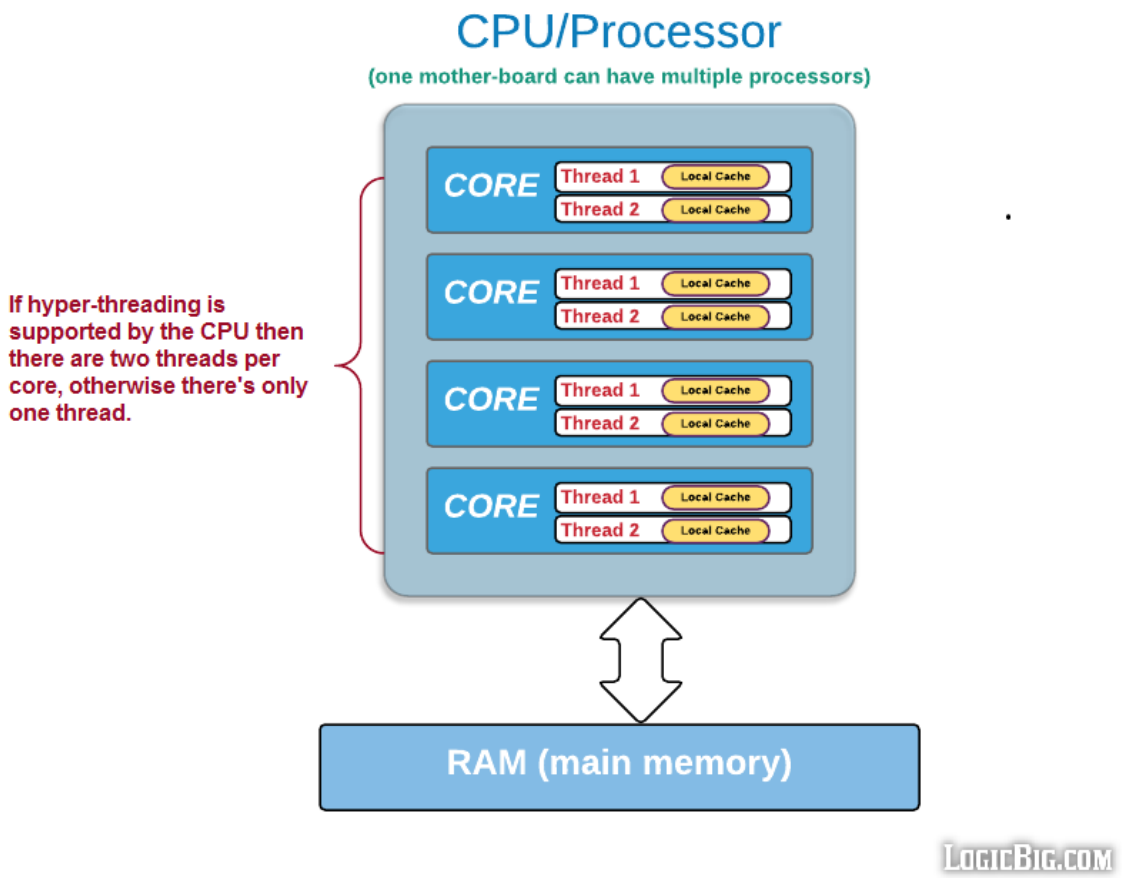
https://scx2.b-cdn.net/gfx/news/hires/2009/armdeliverst.jpg

Wielordzeniowość



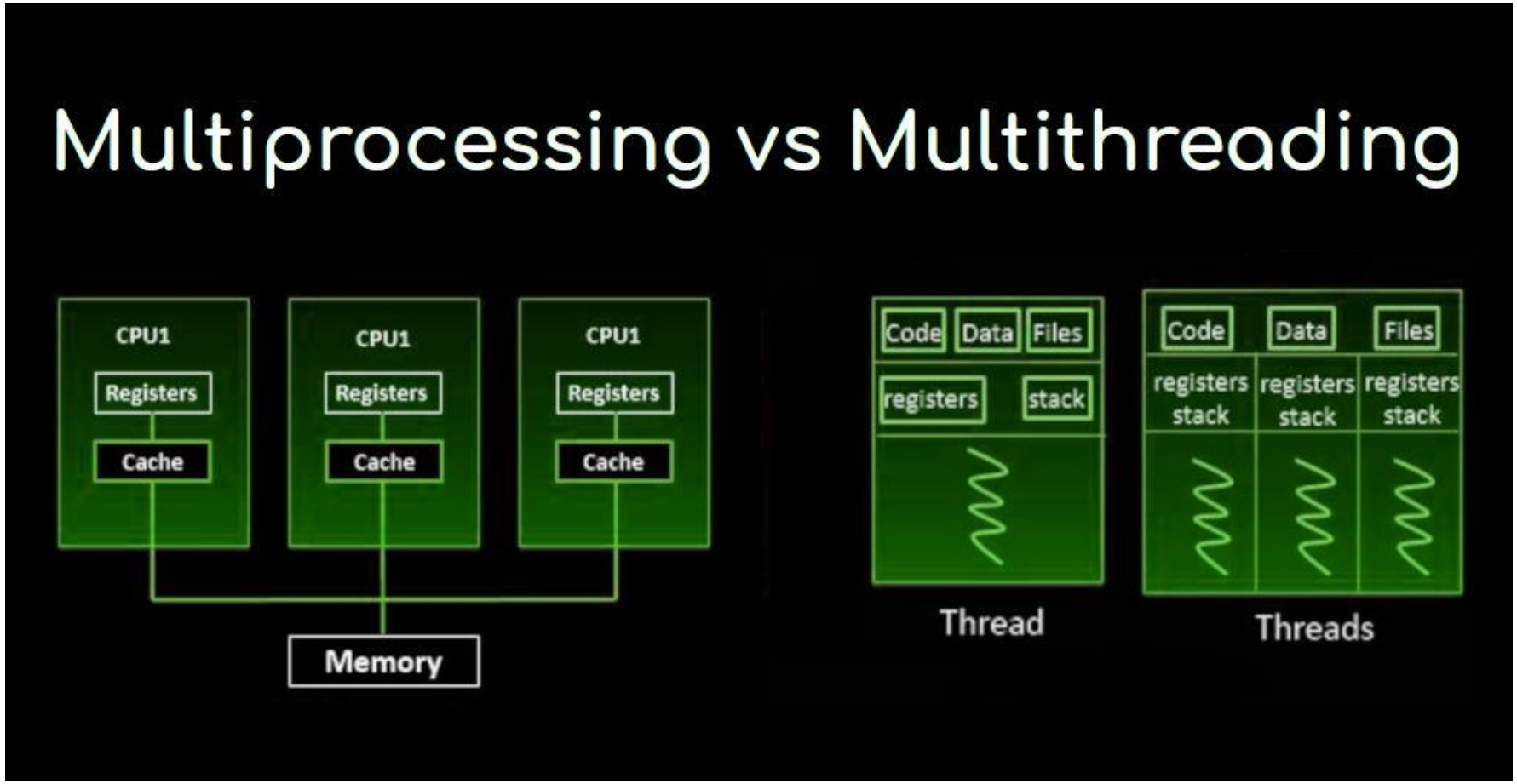
https://cnet3.cbsistatic.com/img/LHigl-O5D8inU7utnj1MPahyI6s=/1200x675/2011/09/13/75b83cad-f0f2-11e2-8c7c-d4ae52e62bcc/inside_intel_sandy_bridge_quad_core_processor.jpg

Wielowątkowość/wielopotokowość



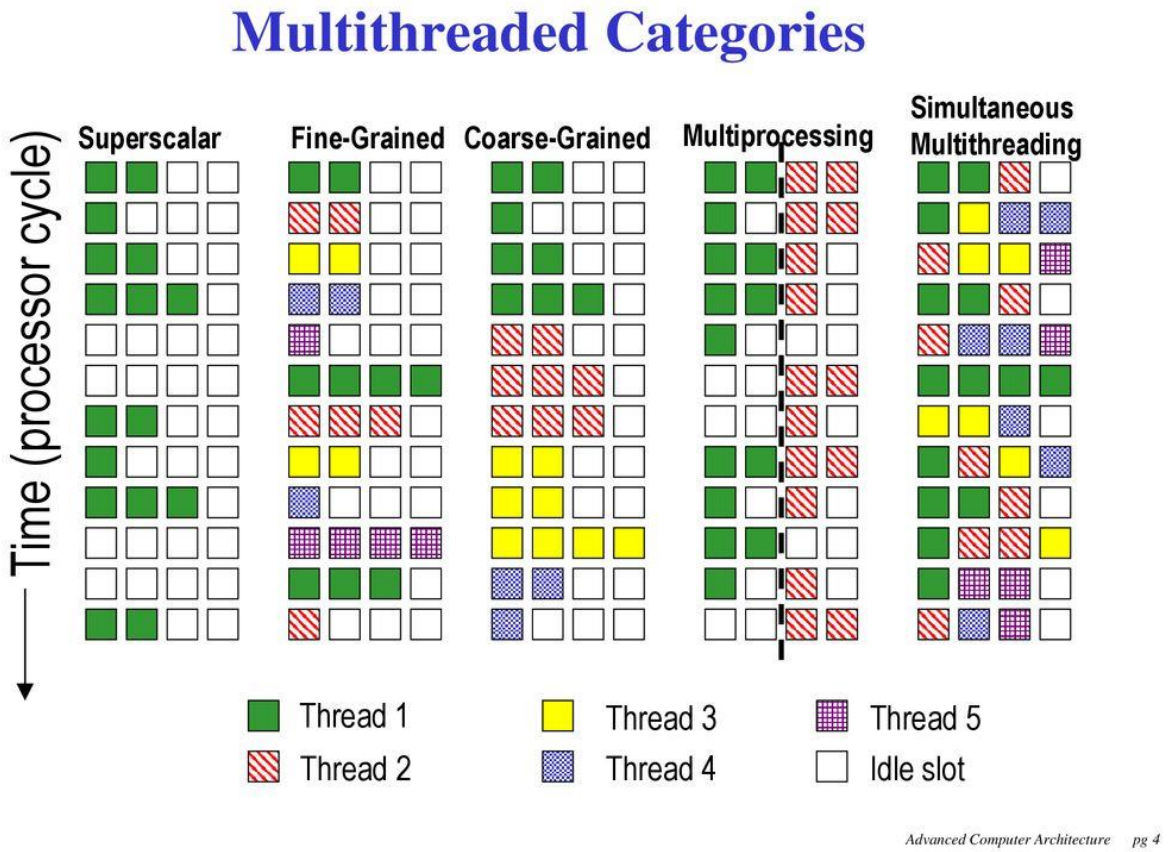
https://www.logicbig.com/quick-info/images/multithreading.png

Wielowątkowość/wielopotokowość



https://i.imgur.com/eO0ux4b.jpg

Wielowątkowość/wielopotokowość



https://slideplayer.com/slide/13872740/85/images/4/Multithreaded+Categories.jpg

Przetwarzanie współbieżne

- Przetwarzanie wielu procesów/wątków, które współdzielą ze sobą dane
- W przypadku jednego rdzenia jednego procesora - przełączanie wątków np. w równych odstępach czasowych
- Środowisko wieloprocesorowe i wielordzeniowe - możliwość wykonywanie operacji równolegle (parallel)
- Z tego rozwiązania korzystają np. aplikacje multimedialne (gry), aplikacje serwerowe (serwer WWW, streaming, komunikatory)
- Wada rozwiązania - możliwość utraty spójności danych, zakleszczenia, naruszenie bezpieczeństwa przetwarzanych danych (Spectre)

Co to Spectre

- To pojęcie obejmuje dwie wykryte luki - CVE-2017-575 (bounds check bypass) oraz CVE-2017-5715 (branch target injection)
- Zagrożenie na poziomie sprzętowym
- Polega na uruchomieniu w systemie potencjalnie groźnego oprogramowania
- Oprogramowanie uruchomione z uprawnieniami użytkownika może, w ramach swojego wątku, odczytywać dane także z wątku aktualnie wykonywanego w ramach tego samego rdzenia (wskazanie jest także na ten sam procesor)
- Problem wykorzystuje fakt, że poszczególne wątki i procesy współbieżne nie miały separacji pamięci cache.

Programowanie równoległe

- Występuje przy dwóch lub większej ilości niezależnych jednostek przetwarzania w jednym czasie
- Programowanie równoległe nie przekreśla wykonywania współbieżnego
- Sprzęt umożliwiający wykonywanie kodu równolegle:

1) wielordzeniowe procesory

2) wieloprocesorowe układy

3) jednostki przetwarzania grafiki (GPU)

4) field-programmable gate arrays (FPGAs)

5) klastry komputerowe

Więcej o problemach współbieżności

- Interakcja pomiędzy poszczególnymi wątkami/procesami
- Uporządkowanie instrukcji tzw. Atomowych
- Trudność dokonania analizy poprawności programu współbieżnego, jego przebiegu oraz możliwości wystąpienia tzw. Niepożądanego przeplotu (wykonania wątku/procesu, który nie powinien się w danej chwili wykonać)
- Brak precyzyjnej liczby testów, które mogłoby dać jednoznaczną odpowiedź dotyczącą poprawności programu
- Ponadto aplikacje mogą próbować "prześcigiwać się" w dostępie do wspólnego obszaru pamięci danych celem zmiany zapisanych tam zmiennych

Synchronizacja

- Jest jedną z podstawowych metod eliminacji niepożądanych zjawisk niedeterministyczności wątków/procesów
- Oznacza w pewnym stopniu wymuszenie przez programistę kolejności wykonywania się wątków i/lub zatrzymywania ich celem osiągnięcia określonego stanu całego rozwiązania
- Projektant rozwiązania zawsze powinien zakładać niedeterministyczny przebieg programu
- Źle dobrana synchronizacja może zepsuć efekt współbieżności (można doprowadzić do spowolnienia rozwiązania)
- Inne następstwa błędnej synchronizacji – zakleszczenie (kolejne wątki/procesy czekają na dane z innych procesów, zaś aktualnie wykonywane oczekują na rozwiązania aktualnie wstrzymanych)
- Kolejny problem - zagłodzenie (brak dopuszczenia wątku/procesu do wykonania)

Przetwarzanie równoległe - wstęp

- Jeden ze sposobów wykonywania programów przez sprzęt komputerowy dysponującego wieloma jednostkami przetwarzającymi (w środowisku heterogenicznym)
- Głównym zastosowaniem zrównoleglenia były wysokobciążające obliczenia
- Obecnie zrównoleglenia wykorzystuje się przy pracy programów, takich jak przeglądarki WWW, aplikacjach sieciowych, symulacjach środowiskowych czy grach komputerowych
- Ze względu na charakter dokonywanych zrównolegleń możemy wyróżnić dwa typy przetwarzania – scentralizowane i rozproszone

Scentralizowany model przetwarzania równoległego

- Wykonywany przeważnie w ramach jednej maszyny komputerowej
- Zmienne poszczególnych wątków i procesów mogą (i zapewne to robią) współdzielić zmienne pomiędzy sobą
- W tym wypadku właśnie pojawia się najwięcej niepożądanych problemów zrównoleglania programów (synchronizacja, wyścig, zagłodzenie, zakleszczenie)
- Jeżeli programista nie przewidział jednej z takich operacji, w rozwiązaniu może wziąć udział tzw. Arbiter pamięci
- Model ten przedstawiany jest jako trudniejszy do oprogramowania

Rozproszony model przetwarzania równoległego

- Model ten zakłada współpracę wielu maszyn ze sobą, chociaż są wyjątki (działanie kilku procesów w ramach jednej maszyny bez współdzielenia pamięci)
- Brak więc pamięci współdzielonej
- Co za tym idzie – brak możliwości dokonania realnej synchronizacji
- Brak jednoczesnej komunikacji pomiędzy procesami
- Rozwiązaniem nowego problemu jest przesyłanie komunikatów pomiędzy procesami i/lub komunikacja poprzez protokół RPC (Remote Procedure Call)
- Rozwiązanie (niezależnie które) dokładają dodatkowy narzut na program (dodatkowy czas, który trzeba obsłużyć - kolejne wyzwanie)

Działanie asynchroniczne

- Komunikaty przesyłane są wtedy, kiedy nadawca będzie tego potrzebował
- Nadawcy nie interesuje, czy odbiorca komunikatu czeka na wiadomość…
- … nawet czy takowy istnieje
- W przypadku przetwarzania współbieżnego istotne może być utworzenie tzw. Buforu danych, do którego będą trafiać wysyłane przez nadawcę dane, by następnie być wykorzystanymi do określonych operacji
- Jeżeli bufor będzie w danej chwili pełny - można po prostu wstrzymać dalsze operacje wątku/procesu dodającego informacje do bufora (blokowanie) lub zwrócić błąd (wersja nieblokująca)
- Z kolei odbiorca, jeżeli będzie chciał odczytać z bufora nieistniejące dane będzie mógł albo zaczekać na nadejście danych (blokująca) lub zgłosić błąd

Działanie synchroniczne

- Komunikacja odbywa się tylko przy dodatkowych komunikatach informacyjnych
- Odbiorca, jeżeli wymaga od nadawcy danych informacji, czeka na chwilę, aż rzeczony nadawca je wyśle bądź zakomunikuje chęć przesłania danych
- Analogiczne działanie można zaobserwować gdy to nadawca będzie chciał przesłać dane
- Możliwe są różne warianty komunikacji synchronicznej – jeden-do-jednego, jeden-do-wielu, wiele-do-wielu

Przykłady implementacji – MPI (Message Passing Interface)

- Jeden ze najstarszych standardu komunikacji pomiędzyprocesowej
- Ma formę biblioteki dostępnej dla języków C, C++, Fortran
- Istnieje wiele implementacji MPI - zarówno otwartoźródłowe, jak i zamknięte oraz w tzw. Domenie publicznej
- Zaletą jest skalowalność, przenośność oraz unifikacja komunikacji (te same zestawy komunikatów)
- Komunikacja w rozwiązaniu opiera się na: komunikatorach, operacjach punkt-punkt, rozgłaszaniu, redukcji
- Odbierane/wysyłane dane muszą mieć odpowiedni typ danych (ustalony dla MPI)

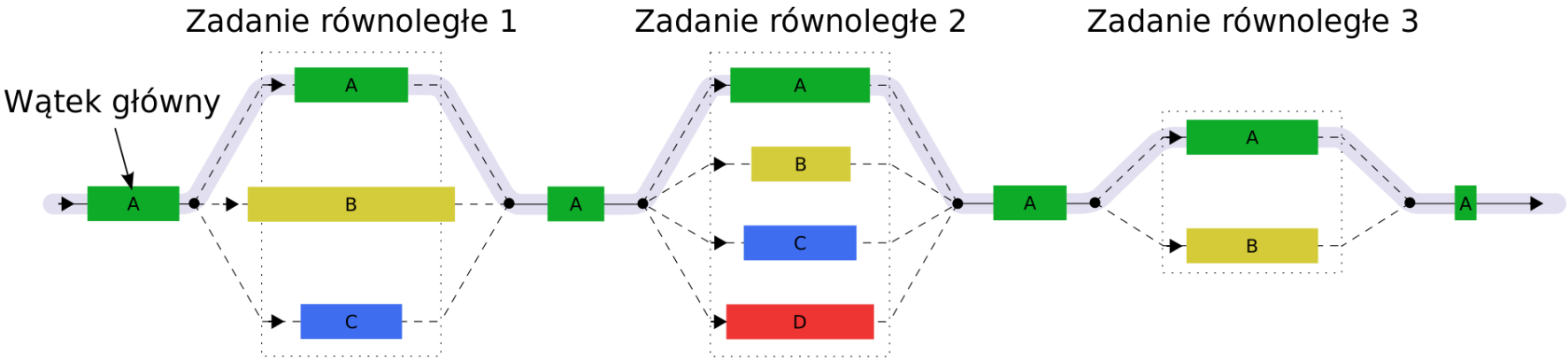
```c
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv)
{
    char buf[256];
    int my_rank, num_procs;

    /* Initialize the infrastructure necessary for communication */
    MPI_Init(&argc, &argv);

    /* Identify this process */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out how many total processes are active */
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

    /* Until this point, all programs have been doing exactly the same.
       Here, we check the rank to distinguish the roles of the programs */
    if (my_rank == 0) {
        int other_rank;
        printf("We have %i processes.\n", num_procs);

        /* Send messages to all other processes */
        for (other_rank = 1; other_rank < num_procs; other_rank++)
        {
            sprintf(buf, "Hello %i!", other_rank);
            MPI_Send(buf, sizeof(buf), MPI_CHAR, other_rank,
                     0, MPI_COMM_WORLD);
        }

        /* Receive messages from all other process */
        for (other_rank = 1; other_rank < num_procs; other_rank++)
        {
            MPI_Recv(buf, sizeof(buf), MPI_CHAR, other_rank,
                     0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s\n", buf);
        }

    } else {

        /* Receive message from process #0 */
        MPI_Recv(buf, sizeof(buf), MPI_CHAR, 0,
                 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        assert(memcmp(buf, "Hello ", 6) == 0);

        /* Send message to process #0 */
        sprintf(buf, "Process %i reporting for duty.", my_rank);
        MPI_Send(buf, sizeof(buf), MPI_CHAR, 0,
                 0, MPI_COMM_WORLD);

    }

    /* Tear down the communication infrastructure */
    MPI_Finalize();
    return 0;
}
```

Przykład OpenMPI

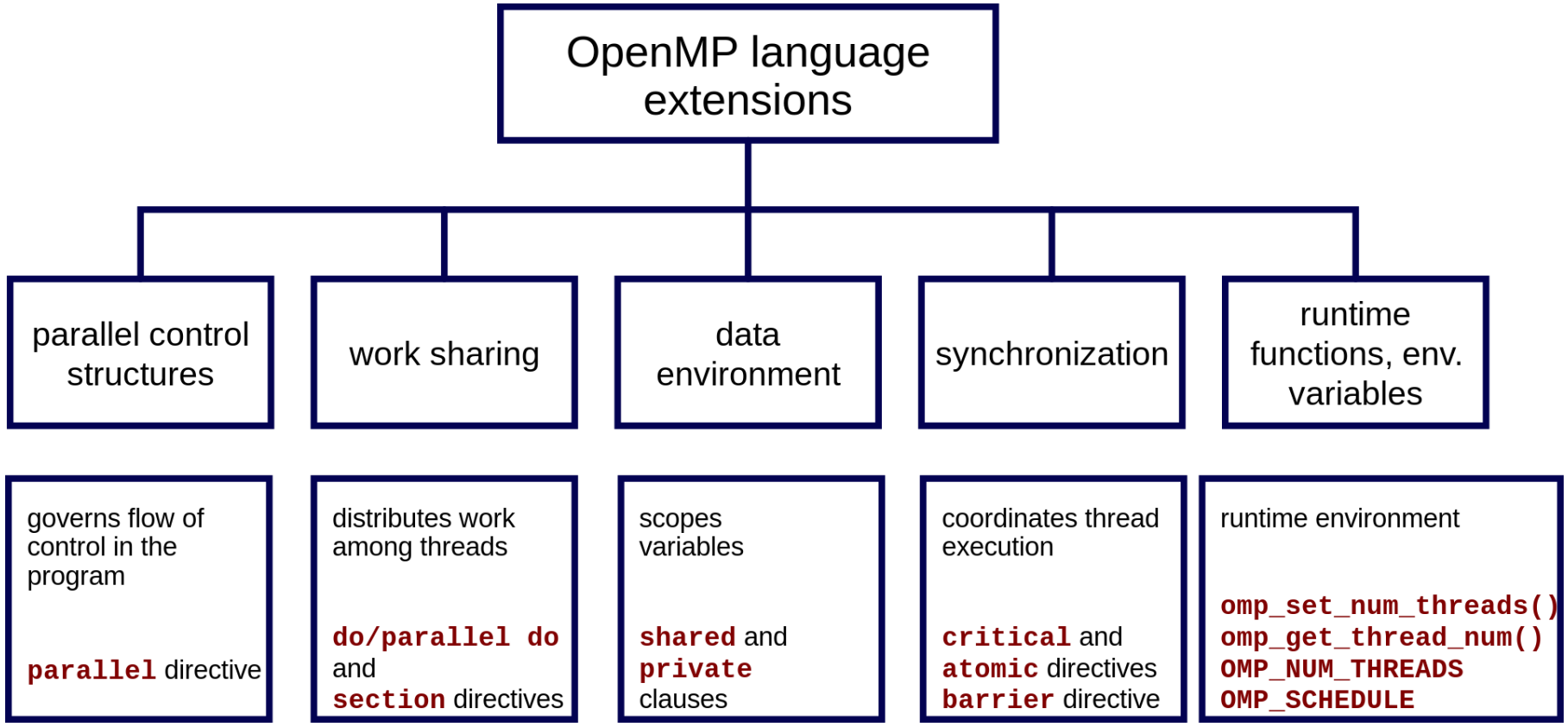
OpenMP

- Kolejny uniwersalny, przenośny standard zrównoleglania
- Głównie do zastosowania scentralizowanego
- Podobnie do MPI - działa dla C, C++, Fortran
- Wymaga dodania do programu biblioteki obsługującej
- Posiada własny typ zmiennych
- Obsługę wielowątkowości obsługuje się poprzez odpowiednie dyrektywy preprocesora (dyrektywy kompilatora)

OpenMP - koncepcja



Konstrukcja OpenMP



OpenMP – prosty kod

```c
int main(int argc, char* argv[])
{
  #pragma omp parallel
  printf("Hello, world!\n");
  return 0;
}
```

OpenMP - zrównoleglenie operacji for

```
int main(int argc, char **argv) {
    const int N = 100000;
    int i, a[N];

    #pragma omp parallel for
    for (i = 0; i < N; i++)
        a[i] = 2 * i;

    return 0;
}
```

OpenMP barrier

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
 int th_id, nthreads;
#pragma omp parallel private(th_id)
 {
  th_id = omp_get_thread_num();
  std::cout << "Hello World from thread" << th_id << "\n";
#pragma omp barrier
 if ( th_id == 0 ) {
   nthreads = omp_get_num_threads();
   std::cout << "There are " << nthreads << " threads\n";
  }
 }
 return 0;
}
```

OpenMP - reduction

```
#define N 10000 /*size of a*/
void calculate(int); /*The function that calculates the elements of a*/
int i;
long w;
long a[N];
calculate(a);
long sum = 0;
/*forks off the threads and starts the work-sharing construct*/
#pragma omp parallel for private(w) reduction(+:sum) schedule(static,1)
for(i = 0; i < N; i++)
   {
     w = i*i;
     sum = sum + w*a[i];
   }
printf("\n %li",sum);
```

OpenMP - critical

```
...
long sum = 0, loc_sum = 0;
/*forks off the threads and starts the work-sharing construct*/
#pragma omp parallel for private(w,loc_sum) schedule(static,1)
{
  for(i = 0; i < N; i++)
    {
      w = i*i;
      loc_sum = loc_sum + w*a[i];
    }
  #pragma omp critical
  sum = sum + loc_sum;
}
printf("\n %li",sum);
```

```c
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

/*defines the total amount of iterations*/
#define N_ITER 1024
int main(void)
{
    int i,id,chunk_size,numit;
    int array[N_ITER];
    memset(array,0,sizeof(array));
#pragma omp parallel default(none) private(i, id, numit) shared(array)
    {
    numit=0;
    id = omp_get_thread_num();
    printf ("Thread no %d starting... \n", id);
    srand(time(0)*(1+id));
#pragma omp for schedule(runtime) private(id)
    for (i = 0; i < N_ITER; ++i)
    {
        id = omp_get_thread_num();
        usleep(rand()%(10000*(1+id)));
        array[i]=id;
        numit++;
    }
    printf("Thread %d performed %d iterations\n",id,numit);
    }
    for (i = 0; i < N_ITER; ++i)
    printf("%d",array[i]);
    puts("");
    return 0;
}
```

OpenMP

Odnośniki i materiały dodatkowe

- https://www.purepc.pl/meltdown-i-spectre-wszystko-o-lukach-w-procesorach-intel-i-amd
- https://www.educative.io/edpresso/what-is-concurrent-programming
- https://pl.wikipedia.org/wiki/Przetwarzanie_wsp%C3%B3%C5%82bie%C5%BCne
- https://takuti.me/note/parallel-vs-concurrent/
- https://www.bestcomputersciencedegrees.com/faq/what-is-parallel-programming/
- https://begriffs.com/posts/2020-03-23-concurrent-programming.html
- https://www.toptal.com/software/introduction-to-concurrent-programming

Odnośniki i materiały dodatkowe

- https://pl.wikipedia.org/wiki/Obliczenia_r%C3%B3wnoleg%C5%82e
- http://smurf.mimuw.edu.pl/book/export/html/1250
- http://www.mif.pg.gda.pl/homepages/marcin/PWIR2018_2019/pwir-1.pdf
- https://en.wikipedia.org/wiki/Message_Passing_Interface#Hardware_implementations
- https://curlie.org/en/Computers/Parallel_Computing/Programming/Libraries/MPI
- https://www.open-mpi.org/software/ompi/v4.1/
- https://pl.wikipedia.org/wiki/OpenMP