

INTEL® XEON PHI™

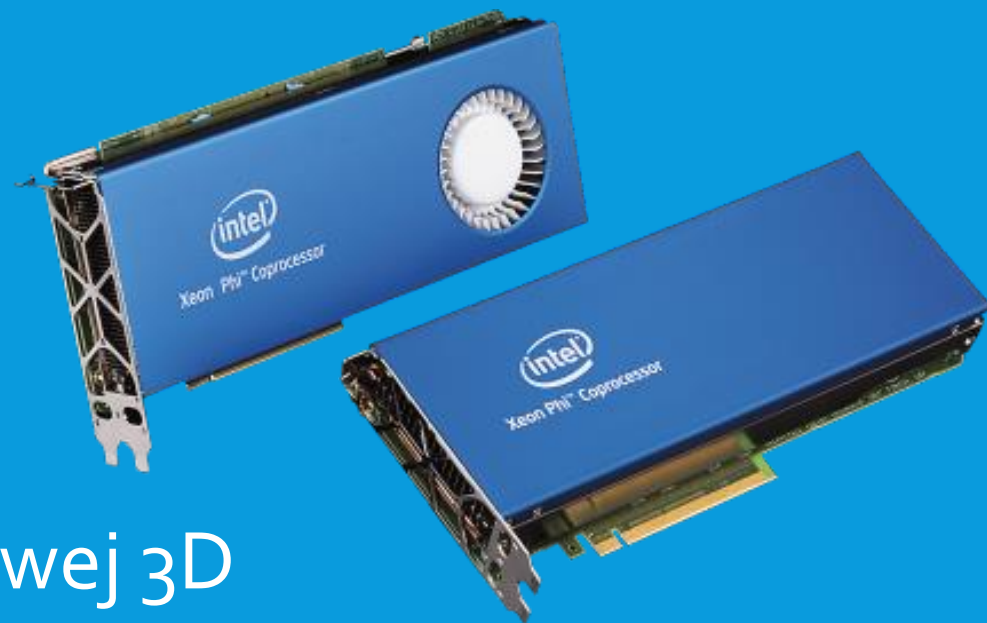
Rodzina koprocessorów architektury Intel MIC

PLAN PREZENTACJI

- Co to jest Intel Xeon Phi
- Gdzie się używa?
- Architektura MIC
- Architektura Software
- Tryby wykonawcze
- Modele programowania
- Programowanie
- Konkretnie zastosowania

C O T O J E S T ?

- Karta rozszerzeń podłączana przez magistralę PCI Express
- Współpracuje z procesorami Intel Xeon
- Zwiększa wydajność wielkoskalowego przetwarzania równoległego kodu
- 1.2 TFLOPS
- Wytwarzane w technologii tranzystorowej 3D Tri-Gate w procesie 22nm



W CZYM LEPSZE/GORSZE OD INTEL XEON?

- więcej rdzeni i wątków
- pojemniejsze wektorowe jednostki wykonawcze (VPU)
- mniejsze częstotliwości rdzeni

Większa łączna wydajność zadań obliczeniowych w procesie przetwarzania równoległego wielkiej skali.

GDZIE WYKORZYSTYWAĆ?

- aplikacje równoległe dużej skali (100+ wątków oprogramowania)
- aplikacje wykorzystujące jednostki wektorowe
- tam gdzie potrzebna większa przepustowość pamięci lokalnej

GDZIE WYKORZYSTYWAĆ?

- Czy są w aplikacji segmenty, które można wykonywać równolegle na ponad 100 wątkach?

- Czy aplikacja jest w stanie wykorzystać 512-bitowe jednostki wektorowe lub zwiększoną przepustowość pamięci lokalnej?

- Intel Xeon Phi

PICK THE RIGHT TOOL FOR THE RIGHT JOB

Parallel and Fast Serial

The right choice for HPC workloads with parallel and serial components



Highly Parallel

Optimized to be the right choice for highly parallel workloads

Programmability Benefits:

Single source, converging ISA • Common environment

APLIKACJE

- astrofizyka: superkomputer COSMOS na Cambridge
- klimat: NASA Overflow, CFSv2
- tworzenie kontentu cyfrowego: Superresolution processing
- finanse: Xcelerit, Monte Carlo RNG
- geofizyka: SPECFEM3D Cartesian, SeisSol
- inżynieria materiałowa: GPAW, VASP
- fizyka: QCDBench

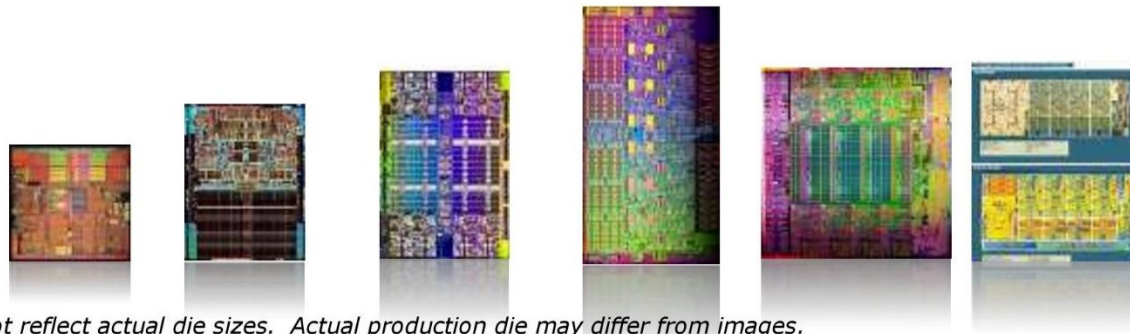
więcej:

<https://software.intel.com/en-us/xeonphionlinecatalog>

ARCHITEKTURA INTEL MANY INTEGRATED CORE

- do 61 rdzeni Intel MIC z taktowaniem 1GHz
- x86 based ISA
- adresowanie 64-bitowe
- 512-bitowe jednostki SIMD
- poza rdzeniami mnóstwo dodatkowych komponentów

ARCHITEKTURA INTEL MANY INTEGRATED CORE



Images do not reflect actual die sizes. Actual production die may differ from images.

	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor E5 Product Family	Intel® Xeon® processor code name Ivy Bridge	Intel® Xeon® processor code name Haswell	Intel® Xeon Phi™ Coprocessor
Core(s)	1	2	4	6	8	10	To be deter mined	61
Threads	2	2	8	12	16	20	To be deter mined	244

Intel® Xeon Phi™ coprocessor extends established CPU architecture and programming concepts to highly parallel applications

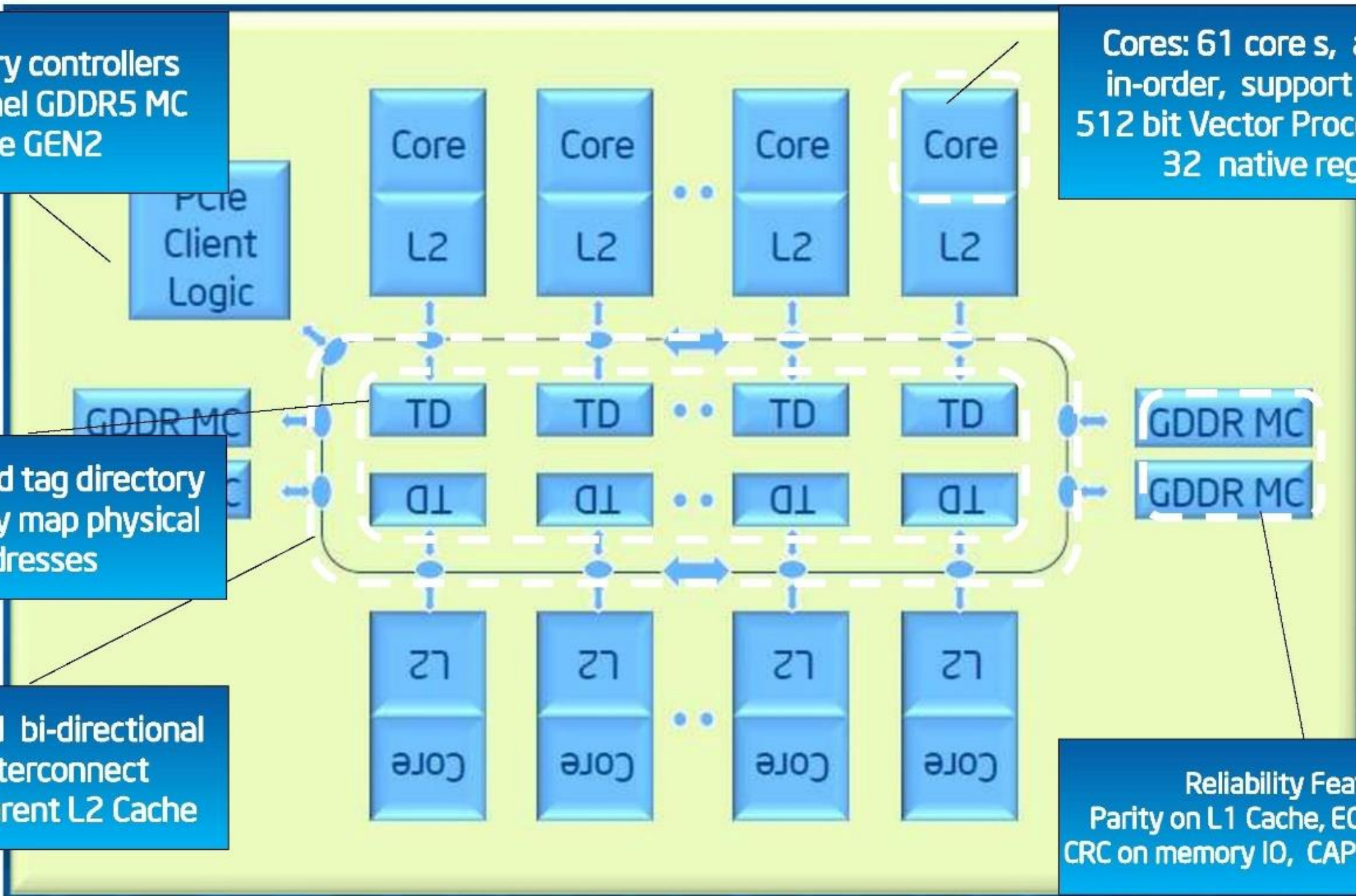
8 memory controllers
16-channel GDDR5 MC
PCIe GEN2

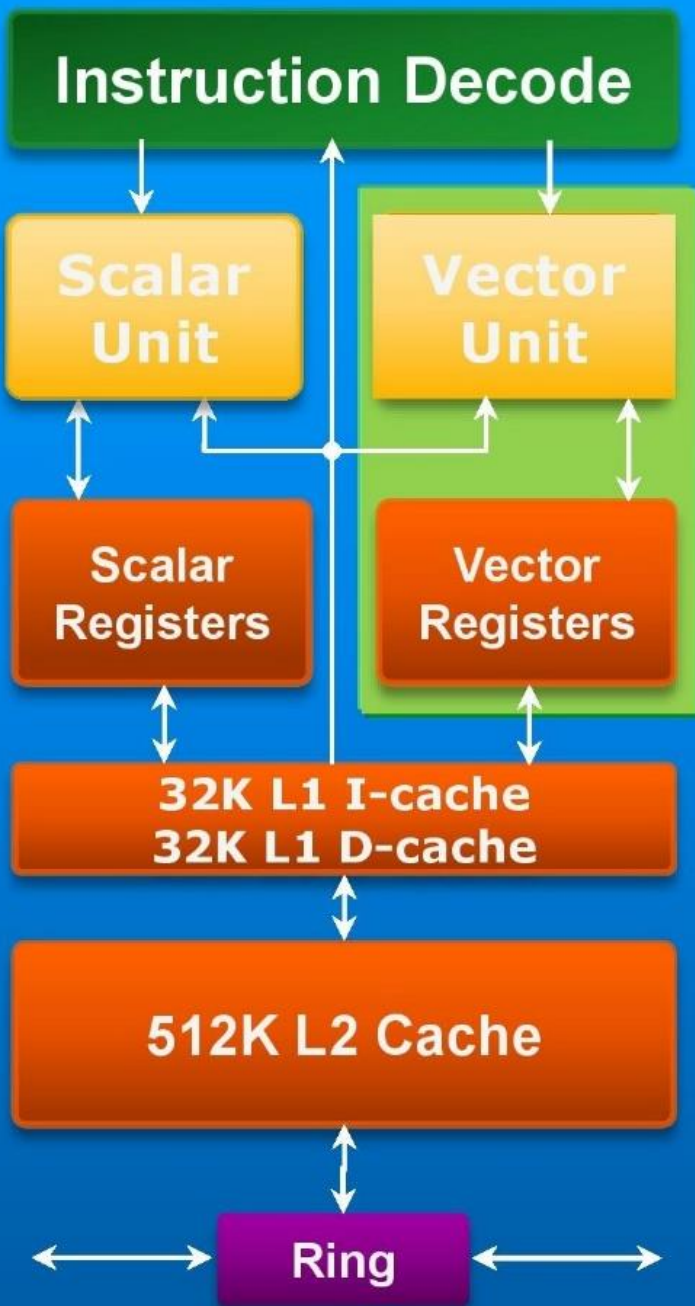
Cores: 61 cores, at 1.1 GHz
in-order, support 4 threads
512 bit Vector Processing Unit
32 native registers

Distributed tag directory
to uniquely map physical
addresses

High-speed bi-directional
ring interconnect
Fully coherent L2 Cache

Reliability Features
Parity on L1 Cache, ECC on memory
CRC on memory IO, CAP on memory IO

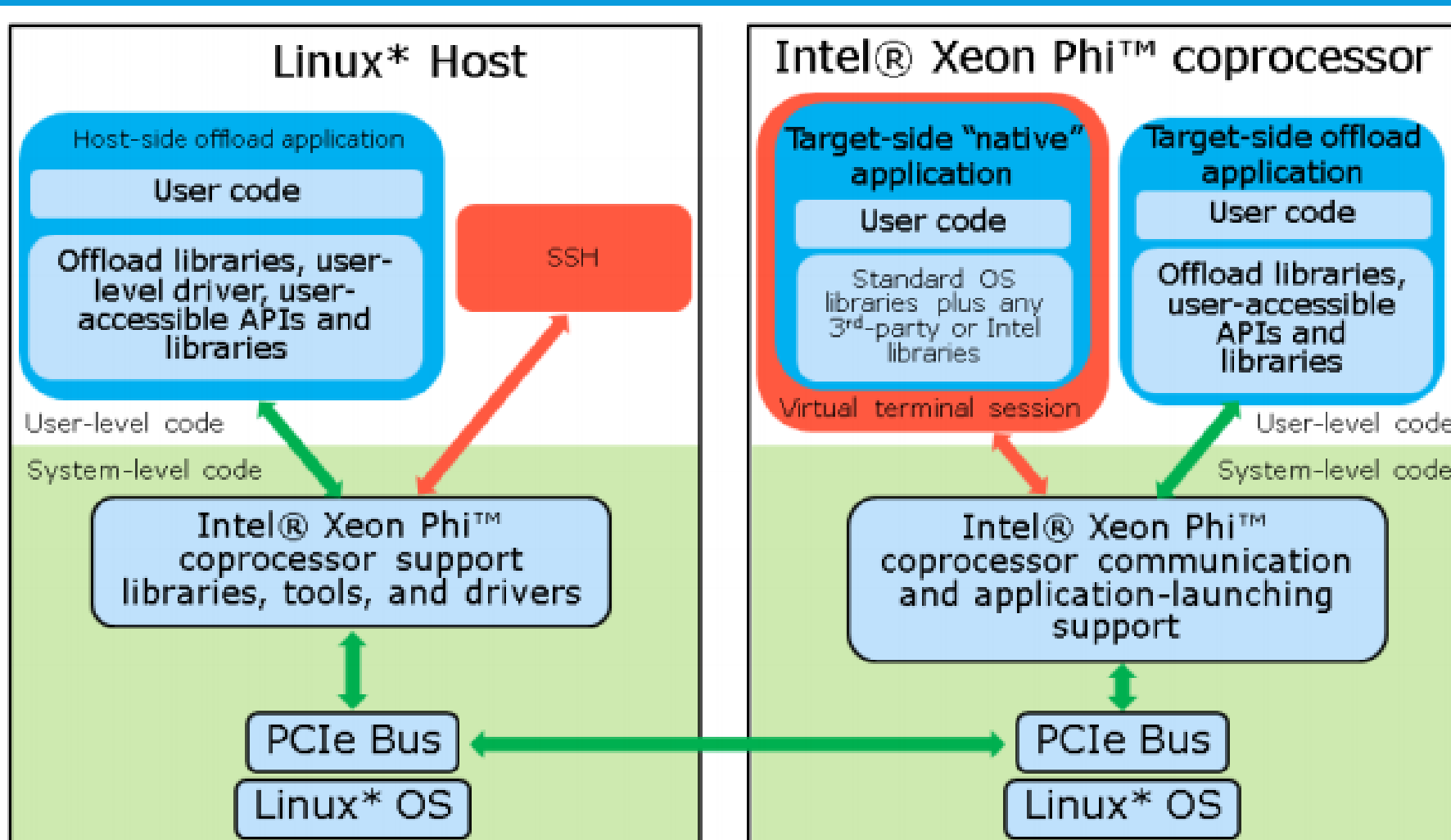




ARCHITEKTURA INTEL MANY INTEGRATED CORE

- Połączone w pierścień rdzenie
- Adresowanie 64-bitowe
- Przetwarzanie potokowe
- 4 wątki fizyczne na rdzeniu
- 512KB L2 Cache

ARCHITEKTURA SOFTWARE



- Linux Inside
- Wsparcie protokołów sieciowych

```
% cat /proc/cpuinfo | head -5
```

```
processor      : 0  
vendor_id     : GenuineIntel  
cpu family    : 11  
model         : 1  
model name    : 0b/01
```

```
%
```

```
% cat /proc/cpuinfo | tail -26
```

```
processor      : 243  
vendor_id     : GenuineIntel  
cpu family    : 11  
model         : 1  
model name    : 0b/01  
stepping      : 1  
cpu MHz       : 1090.908  
cache size    : 512 KB  
physical id   : 0  
siblings      : 244  
core id       : 60  
cpu cores     : 61  
apicid        : 243  
initial apicid : 243  
fpu           : yes  
fpu_exception : yes  
cpuid level   : 4  
wp            : yes  
flags         : fpu vme de pse tsc msr pae mce cx8 apic mtrr mca pat fxsr ht syscall lm lahf_lm  
bogomips      : 2192.10  
clflush size  : 64  
cache_alignment : 64  
address sizes : 40 bits physical, 48 bits virtual  
power management:
```

```
% █
```

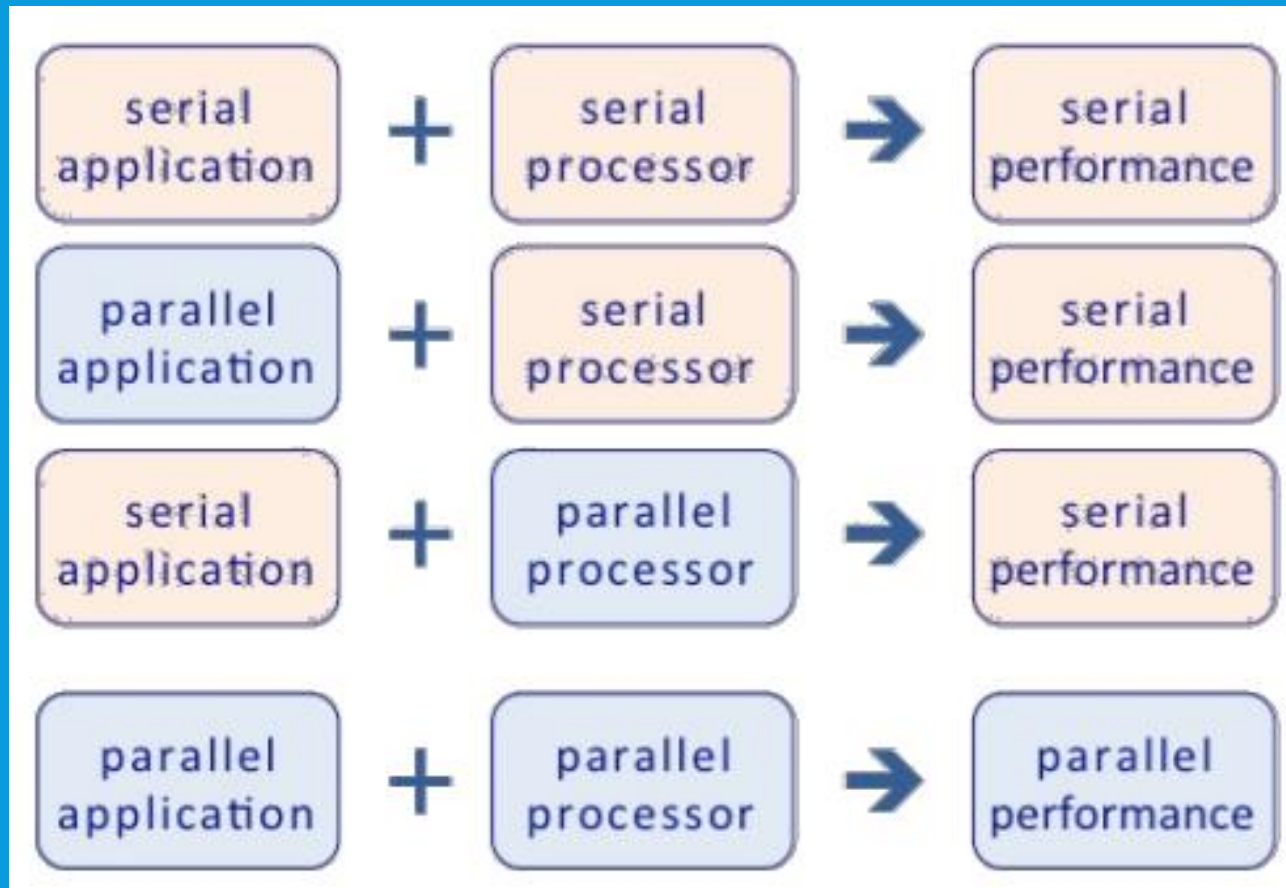
ARCHITEKTURA Z PUNKTU WIDZENIA PROGRAMISTY

- x-86 based SMP-on-a-chip
- 50+ rdzeni
- Wiele wątków na rdzeń
- 512-bitowe instrukcje SIMD
- Instrukcje te same co w 64 bitowym x86 + SIMD
- Wydajne zaawansowane operatory matematyczne
- Działają na Linux

WNIOSEK:

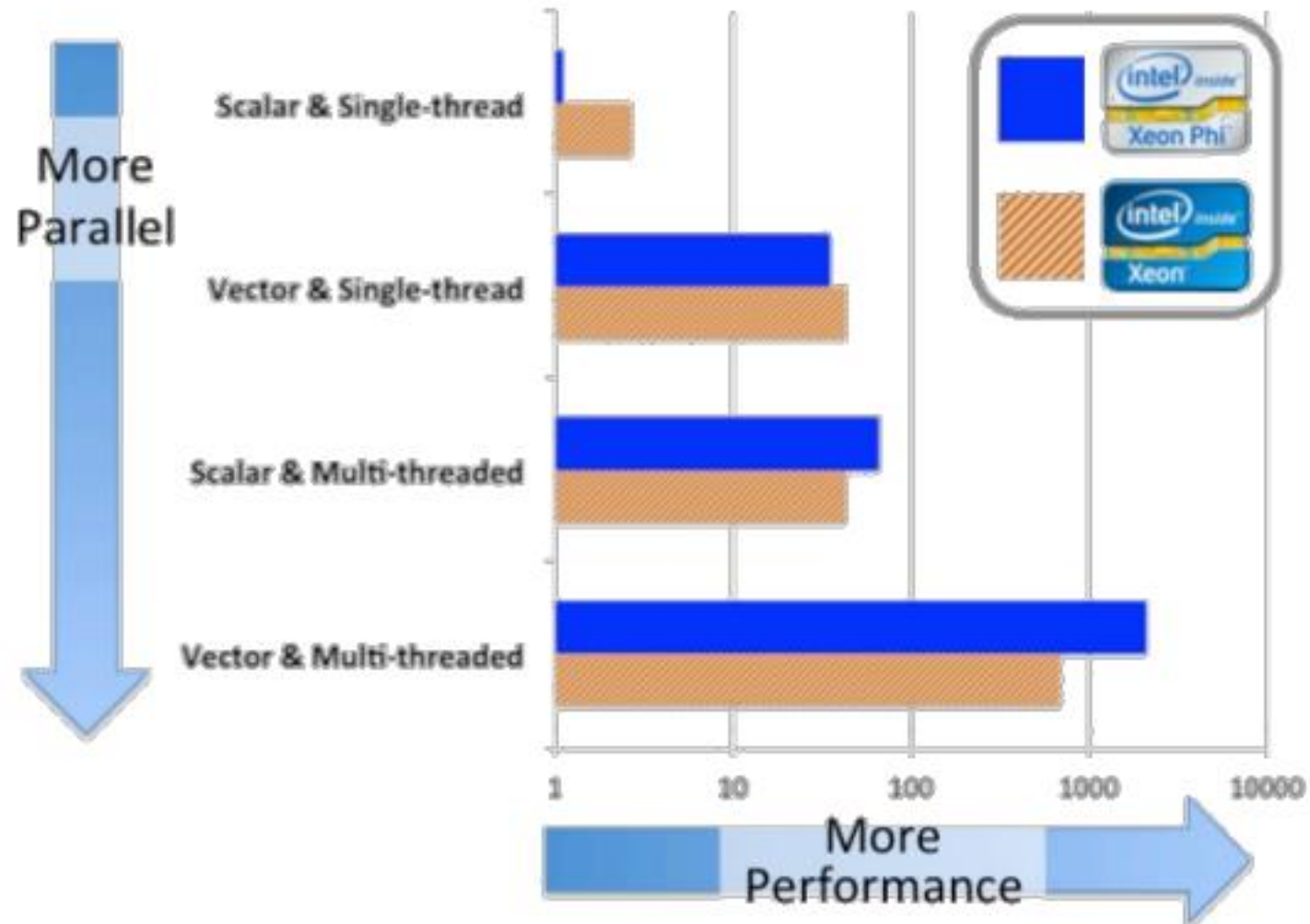
Używaj w tych fragmentach aplikacji, gdzie można znacząco zrównoleglić wykonanie

ARCHITEKTURA Z PUNKTU WIDZENIA PROGRAMISTY

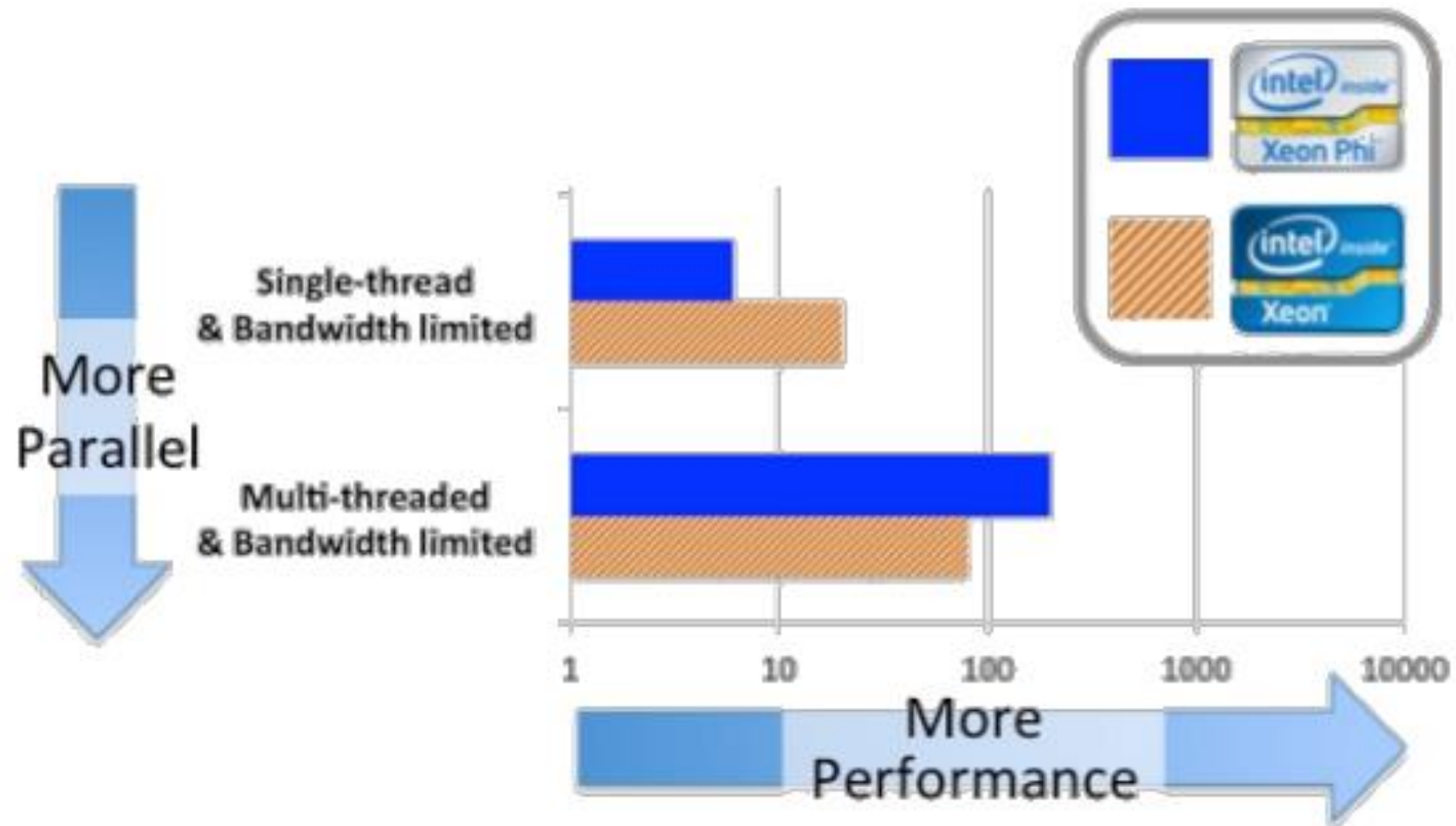


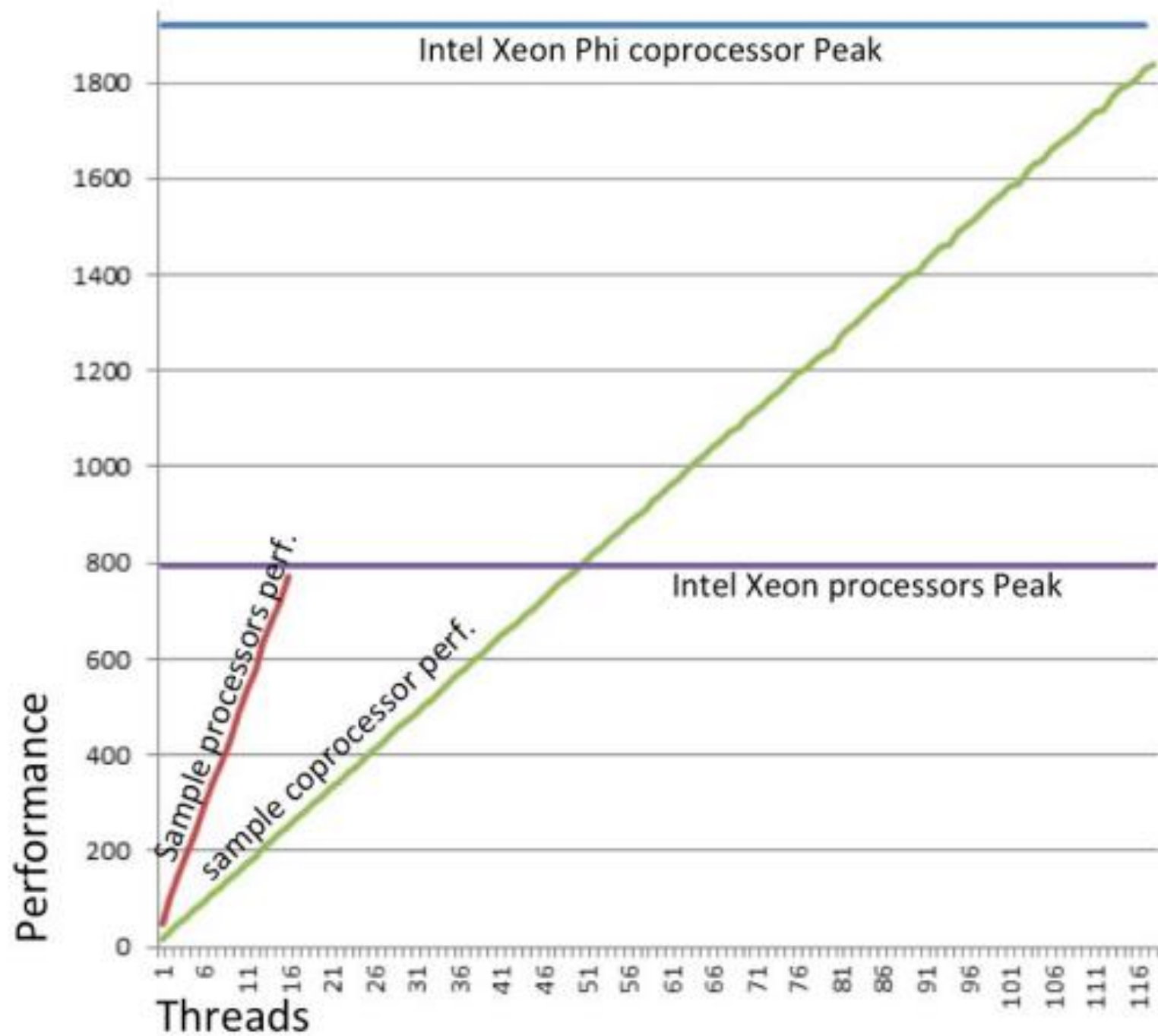
- Najpierw zmaksymalizować użycie hosta Intel Xeon
- Aplikacja może być uznana jako wysoko zrównoleglona jeśli skaluje się na ponad 100 wątków
- Efektywne wykorzystanie wektorów

ARCHITEKTURA Z PUNKTU WIDZENIA PROGRAMISTY



ARCHITEKTURA Z PUNKTU WIDZENIA PROGRAMISTY





ODKRYCIE NA NOWO ZAMIAST PROJEKTU KODU OD PODSTAW

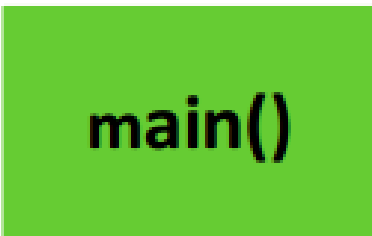
- Aplikacje działają bezproblemowo w ramach wszystkich platform opartych na procesorach Xeon i koprocessorach Xeon Phi => jeden model programowania, nie trzeba przeprojektowywać kodu
- W przeciwieństwie do GPU może obsługiwać system operacyjny z pełną adresowalnością IP i wsparciu standardów
- Wiele trybów wykonawczych

TRYBY WYKONAWCZE

Intel® Xeon
processor

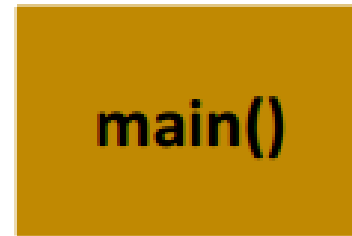


Intel® Xeon Phi™
coprocessor

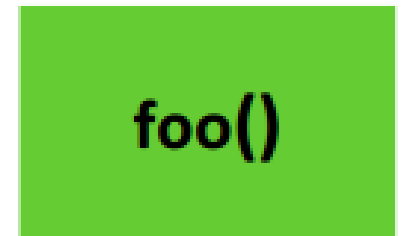


Native

Intel Xeon
processor



Intel Xeon Phi
coprocessor



Offload

NATIVE JEST NAJPROSTSZYM TRYBEM

- Należy tylko skompilować pod architekturę k10m:
 - Intel C/C++ i Fortran compiler
 - Binutils dla k10m
 - LSB – glibc, libm, librt, libcurses...
 - Busybox – minimalne środowisko konsolowe
- Wirtualne sterowniki Ethernet:
 - Ssh, scp
 - NFS mounts

```
[mic@localhost ~]$ cat test.c
#include <stdio.h>
#include <string.h>

#define TWO_MB 2 * 1024 * 1024

int
main()
{
    char *p;

    p = malloc(TWO_MB);
    if (p == NULL) {
        printf("malloc failed!?! \n");
        return (-1);
    }


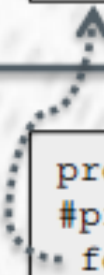
    printf("malloc returned (%p) \n", p);

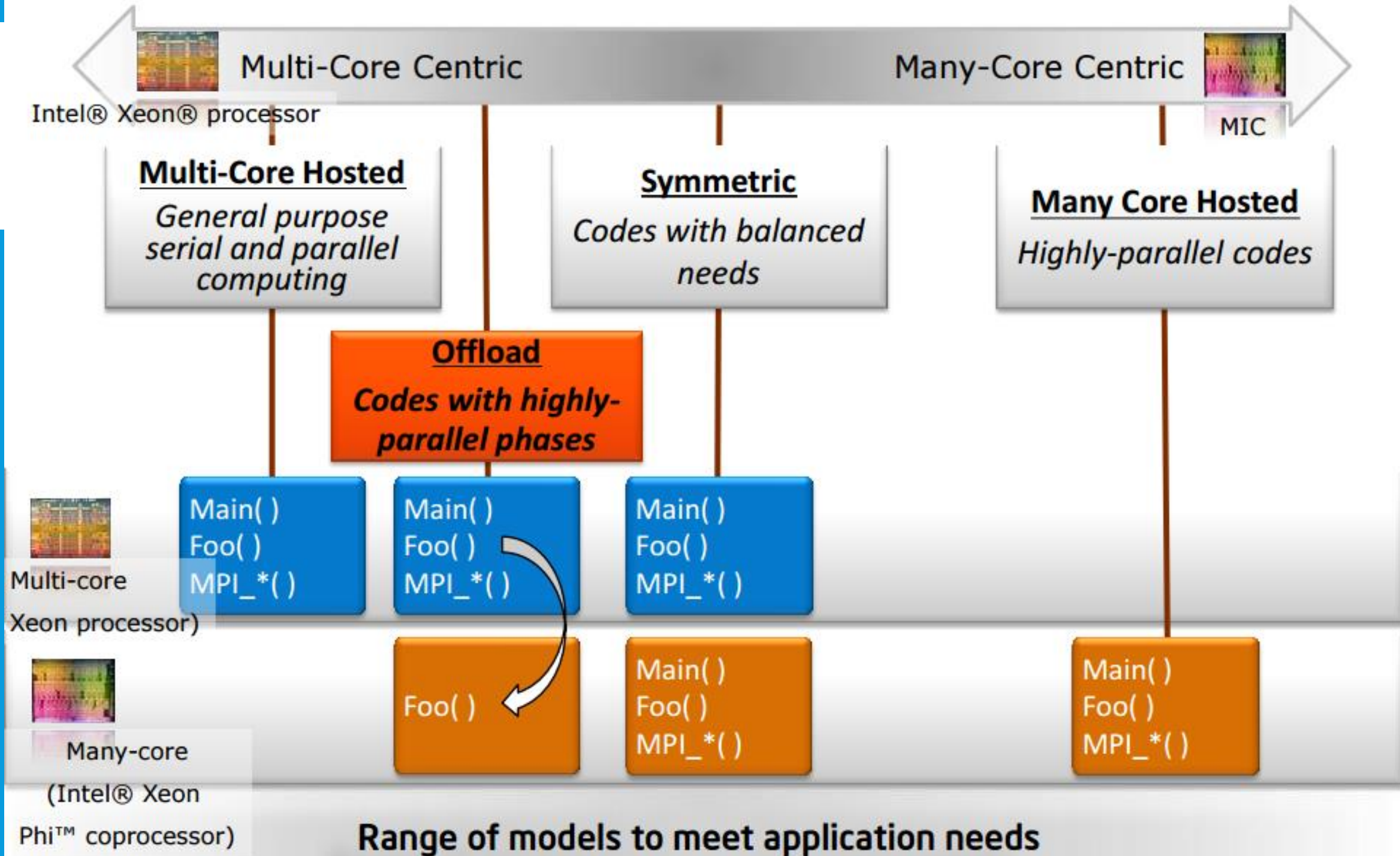
    memset(p, 'a', TWO_MB);
    return (0);
}
[mic@localhost ~]$ icc -mmic test.c -o test
[mic@localhost ~]$ scp test mic0:
test
[mic@localhost ~]$ ssh mic0
[mic@mic0 mic]$ uname -a
Linux mic0.local 2.6.34.11-g65c0cd9 #2 SMP Tue Dec 11 14:34:04 PST 2012 k1om k
[mic@mic0 mic]$ ./test
malloc returned (0x7fa6ad144010)
[mic@mic0 mic]$
```

- Prosty program alokujący 2MB pamięci i wpisujący tam litery a
- Kompilacja do kodu test
- Kopiowanie programu do koprocatora
- Łączymy się z koprocetorem
- Odpalamy program

Host-centric

MIC-centric

	Host Native	Offload	Symmetric	Reverse Offload	MIC Native
Host	<pre>prog() { foo }</pre>	<pre>prog() { #pragma offload foo }</pre> 	<pre>prog() { foo }</pre>	<pre>foo</pre> 	
Phi		<pre>foo</pre>	<pre>prog() { foo }</pre>	<pre>prog() { #pragma offload foo }</pre>	<pre>prog() { foo }</pre>



PĘTLA *FOR* W C TRANSFORMOWANA PRZY UŻYCIU OPENMP

```
#pragma omp parallel for private(j,k)
  for (i=0; i<M; i++) {
    // each thread will work its own part of the problem
    for (j=0; j<N; j++) {
      for (k=0; k<X; k++) {
        // calculation
      }
    }
  }
```

MODELE TRANSFERU DANYCH W TRYBIE OFFLOAD

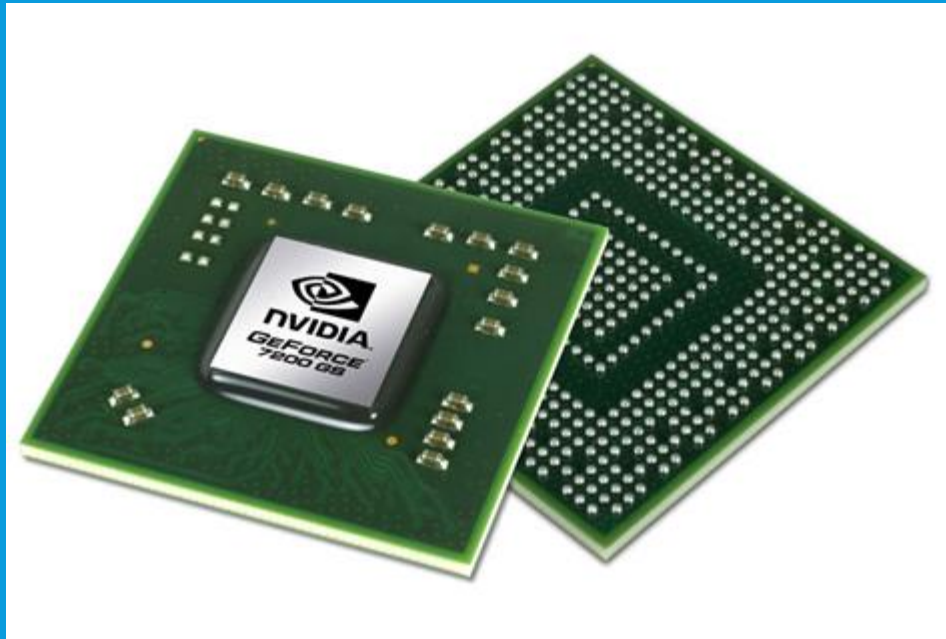
- Explicit Copy
 - Programista oznacza zmienne, które mają być **skopiowane** pomiędzy hostem i kartą
 - Używa do tego dyrektyw offload: pragma/directive-based
 - Przykład:
 - `#pragma offload target(mic) in(data:length(size))`
 - `!dir$ offload target(mic) in(a1:length(size))`
- Implicit Copy
 - Programista oznacza zmienne, które mają być dzielone pomiędzy hostem i kartą
 - Po oznaczeniu tę samą zmienną możemy używać w kodzie hosta i koprocatora
 - Runtime automatycznie utrzymuje spójność zmiennej w programie
 - Przykład:
 - `_Cilk_shared double foo; _Offload func(y);`

POMIAR JAKOŚCI ZRÓWNOLEGENIA

- Sprawdzenie skalowalności: pomiar zmian wydajności przy zwiększaniu liczby wątków
- Sprawdzenie wektoryzacji: kompilowanie z wyłączeniem automatycznej wektoryzacji i sprawdzenie jaki to ma wpływ na szybkość aplikacji
- Weryfikacja użycia cache przez program i lokalności referencji do zmiennych: VTune Amplifier XE

GPU VS INTEL XEON PHI

- Aplikacje dobrze działające na GPU będą dobrze działały na Intel Xeon Phi
- W drugim kierunku niekoniecznie. W szczególności niektóre aplikacje **nie zadziałają na GPU** bez dużych zmian!
- GPU za bardzo różni się od procesora => niemożliwe przedstawione tutaj transformacje



OPTYMALIZACJA UŻYTKOWNIKA

- Transformacje pętli i dostępu do pamięci
- Zagęszczanie wektorów – dane w pamięci są ciągłe
- Użycie całej długości wektorów
- Użycie dużych stron (libhugegetlbf)
- Dobór algorytmów wspierających zrównoleglenie i wektoryzację

MODELE KOMPILACJI I PROGRAMOWANIA

- Nie ma języka specjalnie zaprojektowanego do wykonania równoległego. Najpopularniejsze:
 - Fortran (wbudowane mechanizmy np. DO CONCURRENT, OpenMP, MPI)
 - C (OpenMP, Intel Cilk Plus)
 - C++ (Intel Threading Building Blocks, Intel Cilk Plus, OpenMP, OpenCL)

PROGRAMOWANIE PROCESORA XEON I KOPROCESORA XEON PHI

- Procesory jednordzeniowe to mniejszość na świecie
- Wielordzeniowe procesory to przyszłość => równoległe obliczanie => równoległe programowanie
- Ewolucja metod => wzrost popularności TBB i OpenCL
- Intel wprowadza procesory many-core
- Dodatkowe możliwości uzyskiwane przy użyciu tych samych narzędzi, paradygmatów i języków programowania co w przypadku multicorów

BIBLIOTEKI

- Nie trzeba wymyślać wszystkiego od nowa!
- Intel® Math Kernel Library
- API w znanych językach: OpenMP (C, Fortran), ITBB (C++) oraz MPI (C, C++, Fortran)
- Natywne możliwości systemu (POSIX threads, Windows threads)

OPENMP

```
#pragma omp parallel for reduction(+: s)  
for (int i = 0; i < n; i++)  
    s += x[i];
```

- Standard zaproponowany w 1996
- Każdy duży kompilator C, C++ i Fortrana wspiera OpenMP
- Najczęściej używany w programach naukowych napisanych w Fortran i C

INTEL TBB

```
parallel_for (o, n,  
    [=](int i) {  
        Foo(a[i]);  
    });
```

- Projekt Open Source rozpoczęty w 2006
- Obecnie popularniejszy od OpenMP
- Nie ma konkurencji dla C++

MPI

- Message Passing Interface
- Protokół komunikacyjny będący standardem przesyłania komunikatów pomiędzy procesami równoległymi
- Najpopularniejsza metoda programowania w klastrach obliczeniowych
- Nie ma potrzeby refaktoringu działającego już programu w MPI by działał na Xeon Phi
- Intel® MPI library

INTEL® CILK™ PLUS

```
int fib (int n) {  
    if (n < 2) return 1;  
    else {  
        int x, y;  
        x = cilk_spawn fib(n-1);  
        y = fib(n-2);  
        cilk_sync;  
        return x + y;  
    }  
}
```

Wprowadzone w 2010 rozszerzenie do C/C++
do równoległych zadań

```
cilk_for (int i=0; i<n; ++i){  
    Foo(a[i]);  
}
```

OFFLOADING JESZCZE RAZ

- Dwa sposoby dzielenia danych
 - Przestrzenie pamięci dla poszczególnych rdzeni traktowane oddzielnie =>
Użycie dyrektyw offload do przenoszenia danych i kontroli do koprocesor
- Iluzja współdzielonej pamięci: MYO – Mine Yours Ours

OFFLOADING JESZCZE RAZ

- Użycie pragma offload

```
float *a, *b; float *c;
```

```
#pragma offload target(MIC1)
```

```
    in(a, b : length(s))
```

```
    out(c : length(s) alloc_if(o))
```

```
for (i=0; i<s; i++) {
```

```
    c[i] = a[i] + b[i];
```

```
}
```

OFFLOADING JESZCZE RAZ

- Wskaźniki danych mogą być współdzielone korzystając z bibliotek MYO
- Keyword: `_Cilk_shared`
- Offloading: `x = _Offload func(y);`

COSMOS

- Kod symulujący ewolucję domain walls we wczesnym Wszechświecie
- OpenMP
- Zmiana kodu na cache-friendly, wektoryzacja, zmniejszenie zajętości pamięci
- Stworzenie eigensolvera
- Efekt: 5x Xeon, 17x Xeon Phi



NEC SUPERRESOLUTION PROCESSING

- Konwersja SD -> HD
- Na serwerach NEC przed migracją film 30fps konwertowany z prędkością 6fps
- Intel® C++ Composer XE
- Efekt: realtime conversion

