EXASCALE COMPUTING

Jacek Migdał (jacek@migdal.pl)

JAK BĘDZIEMY PROGRAMOWAĆ KOMPUTERY O WYDAJNOŚCI 10^18 FLOP/S?

Prognoza według

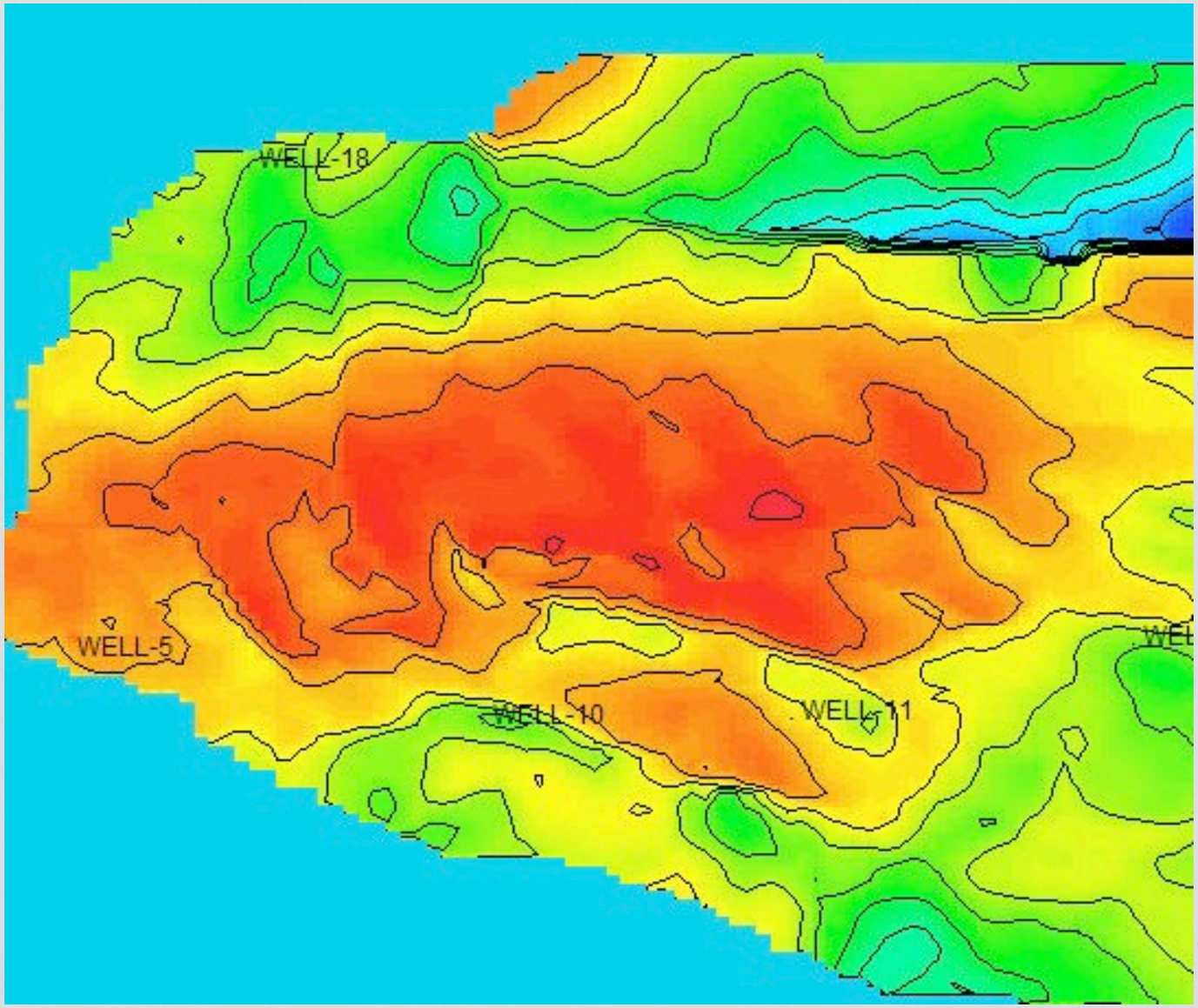
* “The Landscape of Parallel Computing Research: A View from Berkeley“ Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams and Katherine A. Yelick December 18, 2006

* “”ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems” Peter Kogge, Editor & Study Lead, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick September 28, 2008

High-Performace Computing

Obliczenia naukowe, gdzie moc obecnych komputerów jest wielokrotnie niższa od zapotrzebowań.

Przykład:
Szukanie złóż ropy



Przykład:
Szukanie złóż ropy

* szukamy złóż coraz głębiej, szukanie jest coraz droższe
* wiercenie poziome, zmniejszanie napięcia powierzchniowego
* niezwykle przydatne są mapy pokładów (Reverse time migration)

Moc obliczeniowa

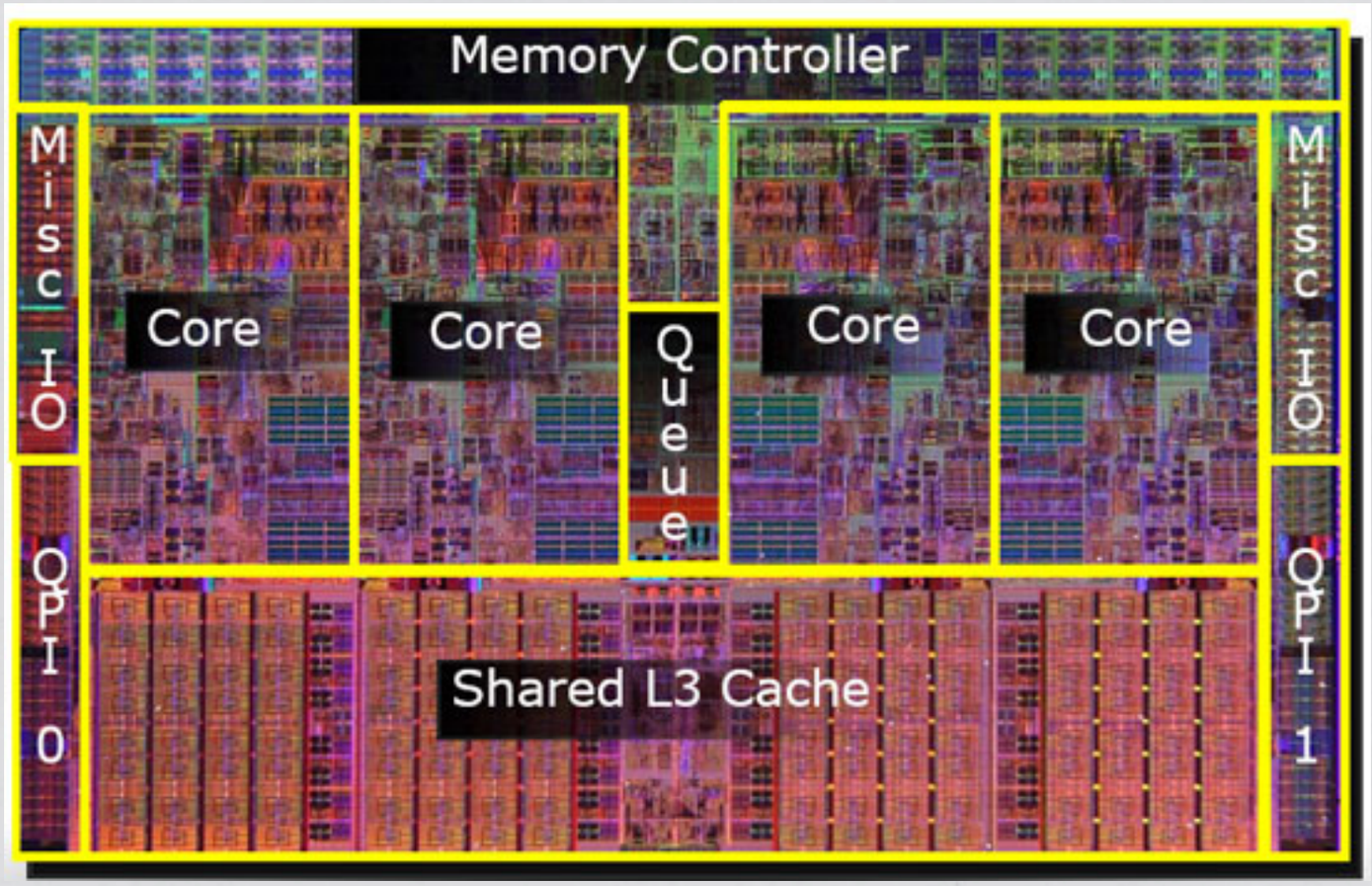
Rok	Wydajność	Maszyna	Liczba rdzeni
1988	> 10^9 Gigaflop/s	Cray YM	8
1998	> 10^12 Teraflop/s	Cray T3E	1480
2008	> 10^15 Petaflop/s	Cray XT5	> 150,000
2018 (?)	> 10^18 Exaflop/s	???	~ 10^9 (?)

Dlaczego jest to ważne?

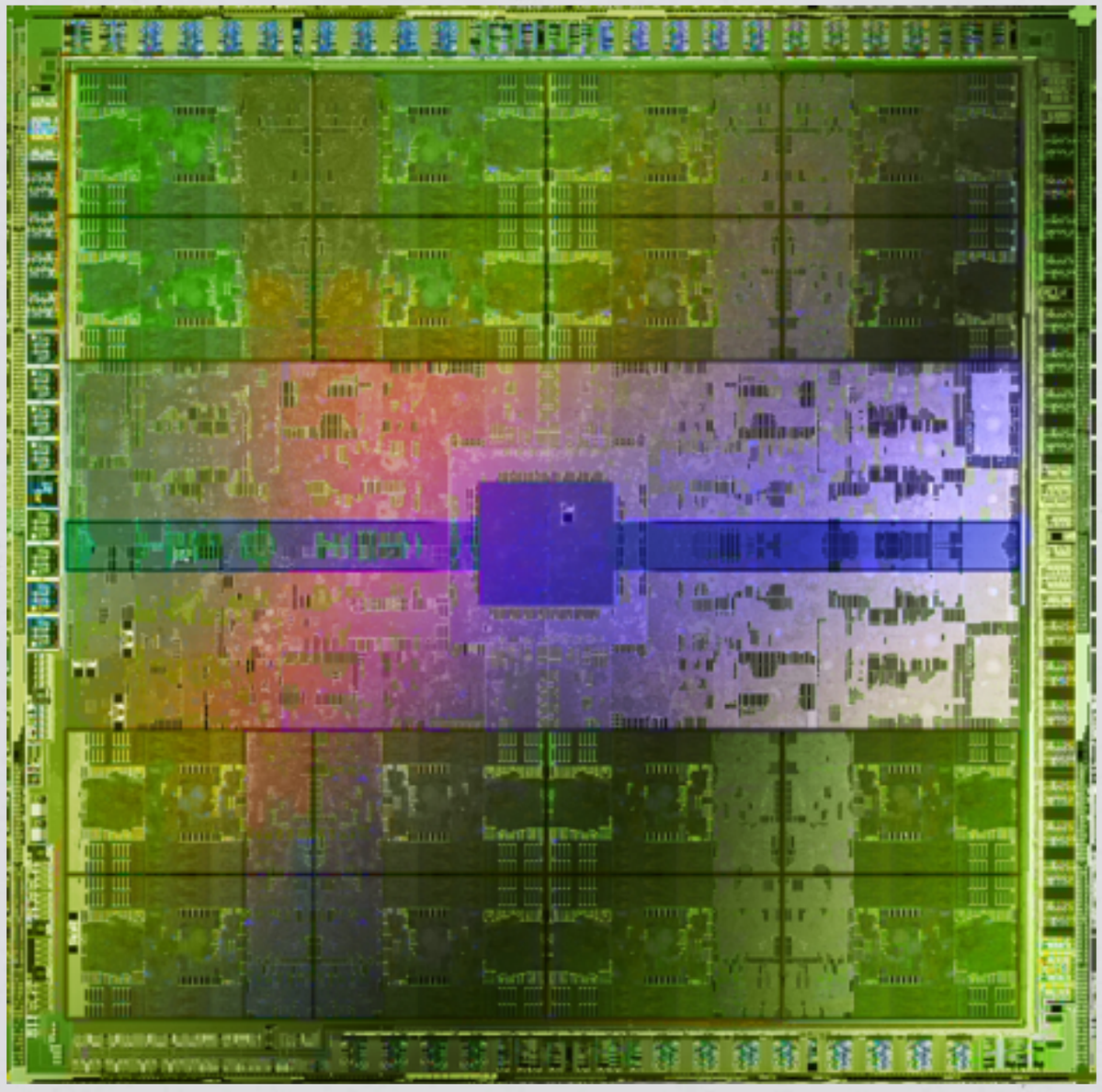
* Pojedynczy rdzeń nie będzie już (znacząco) szybszy, wzrost wydajności będzie wynikał z większego stopnia zrównoleglania.
* Zachowanie obecnego modelu procesorów (x86 kompatybilne wstecz, z dużym cache) będzie trudne.
* Model programowania będzie inny.

PROBLEMY Z OBECNĄ ARCHITEKTURĄ

Intel Core i7



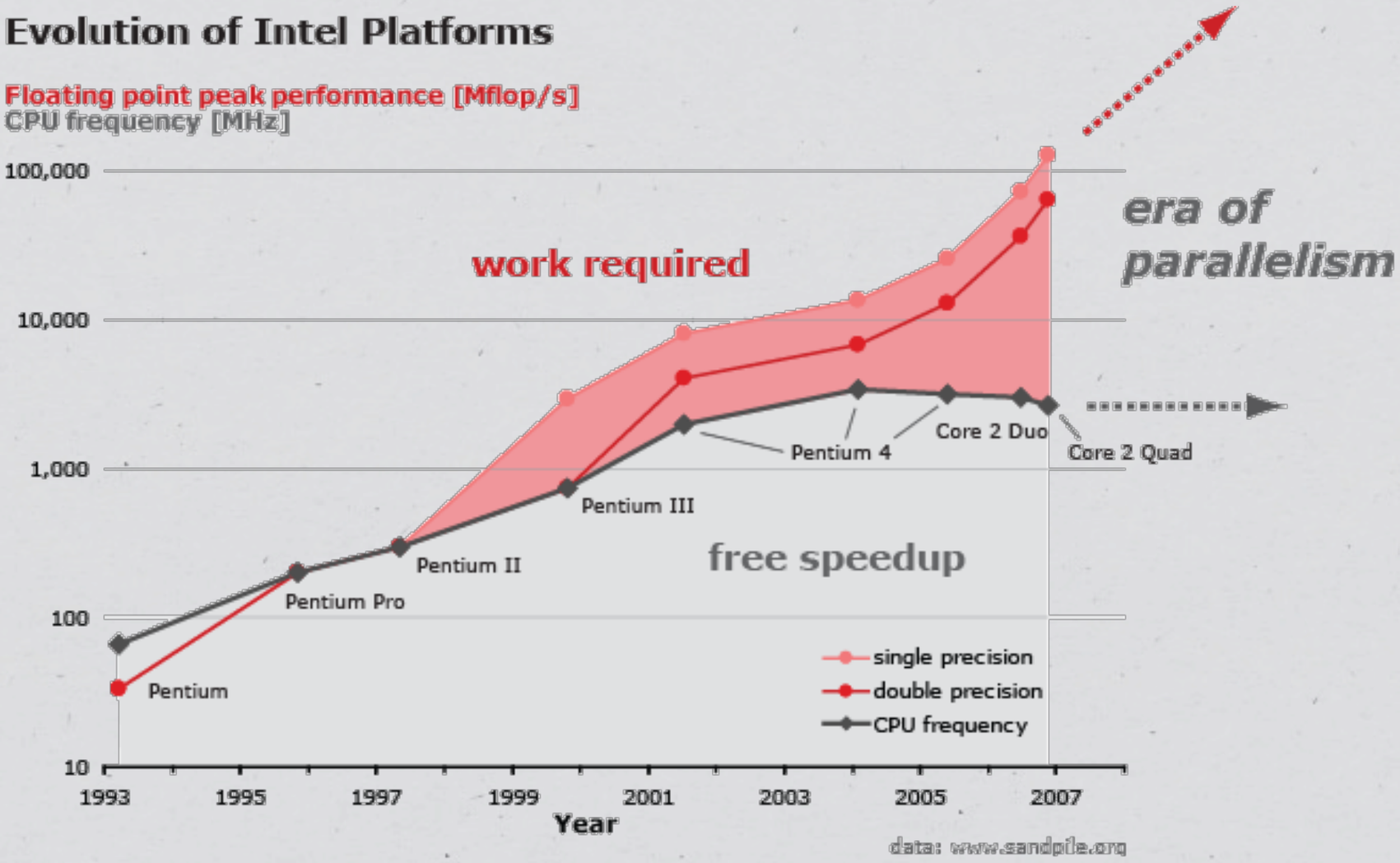
NVIDIA GF100



Power wall

* Do roku 2004 wraz z mniejszym procesem zmniejszało się napięcie. Później liczba elektronów nie pozwala jej zmniejszyć.
* Zwiększanie częstotliwości przy stałym napięciu zwiększa moc.
* Większa moc => więcej ciepła. Problemy z chłodzeniem.

Power wall


Evolution of Intel Platforms
Floating point peak performance [Mflop/s]
CPU frequency [MHz]
100,000
10,000
1,000
100
10
work required
era of parallelism
free speedup
Pentium
Pentium Pro
Pentium II
Pentium III
Pentium 4
Core 2 Duo
Core 2 Quad
single precision
double precision
CPU frequency
1993
1995
1997
1999
2001
2003
2005
2007
Year
data: www.sandpile.org

Najdroższe jest przemieszczanie danych

* szacunki NVIDIA dla procesu 28 nm:
  * 20 pJ, mnożenie DP 64
  * 1 nJ = 1000PJ, przesłanie 64 bitów o 20 mm
* koszt energetyczny obliczeń arytmetyki spada wraz ze zmniejszeniem procesu
* Load/Store są drogie, mnożenie jest tanie

Instruction level parallelism

* wykonywanie wielu instrukcji na raz, np.:
    * out-of order execution
    * branch prediction
* przyśpieszenie bez pracy programisty, ale ma ograniczony potencjał
* utrudnia wykorzystanie pełnej wydajności

Memory wall

* dostarczanie danych do procesora:
  * przepustowość (techniczny problem)
  * opóźnienia (ograniczenia fizyczne)
* mnożenie 4 cykle, dostęp do pamięci RAM ~ 200 cykli

Memory wall

* CPU
  * istotne są opóźnienia
    * wielopoziomowy cache
* GPU
  * istotna jest przepustowość
  * wiele wątków maskuje opóźnienia

Inne problemy

* błędy w pamięci RAM => ECC
* uzysk w procesie produkcji (yield) => modularne procesory
* mało wydajne połączenie RAM z CPU
* błędy w architektury CPU i GPU są niesamowicie drogie => trudno wprowadzać radykalne zmiany

Obecne superkomputery

* na każdym procesorze działa jeden proces (lub k z nich)
* Message Passing Interface (MPI)
* problem z lokalnością danych, wielowątkowością w ramach maszyny

node	**liczba węzłów**
socket	**komputery per węzeł**
core	**rdzenie per maszyna**
vector	**rdzeń, ilość danych**
instruction	**np. pipeline**

WYZWANIA

Wyzwania

* Na szczęście świat jest z natury równoległy. Problemy obliczeniowe w większości też.
* Prawie wszystkie należą do jednej z 13 kategorii (lub ich złożenia).
* Zamiast liczyć teoretycznej wydajności, lepiej te problemy traktować jako benchmarki.

Problemy obliczeniowe

Problem	Przykład	Częste ograniczenie
1. Dense Linear Algebra	BLAS, LAPACK, MATLAB	Flop/s
2. Sparse Linear Algebra	SpMV, SuperLU	Flop/s, BW
3. Spectral (FFT)	FFT, DSP	opóźnienia
4. N-Body	systemy cząsteczkowe	Flop/s

Problemy obliczeniowe

Problem	Przykład	Częste ograniczenie
5. Structured Grid	automaty komórkowe	BW
6. Unstructured Grid		opóźnienia
7. Monte Carlo	wycena opcji	
8. Combinational Logic	szyfrowanie, CRC	BW lub op/s

Problemy obliczeniowe

Problem	Przykład	Częste ograniczenie
9. Graph traversal	BFS, DFS	opóźnienia
10. Dynamic programming	odległość edycyjna	opóźnienia
11. Backtrack and Branch+Bound		?
12. Construct Graphical Models	sieci Bayesa	?

Problemy obliczeniowe

Problem	Przykład	Ograniczenie
13. Finite State Machine	kompilator	trudny do zrównoleglenia

ROZWIĄZANIA

SIMD

* instrukcje wektorowe:
    * np. SSE: 4 mnożenia na raz
* multiprocesory na GPU:
    * 16 podprocesorów, które muszą wykonywać ten sam kod
* prosta metoda, choć ma ograniczenia

Przykład: procesory wektrowe Cray

* 32, 64, 128 operacji arytmetycznych w ramach jednej instrukcji
* mimo doskonałego kompilatora, model programowania był trudny i miał wiele ograniczeń
* powszechne w latach 1970, 1980
* wyparte przez x86 SSE

Multicore vs. Manycore

* Jak najwydajniejszy rdzeń N razy na jednym chipie:
  * CPU: Dual/Quad Core, ... (w przyszłości 10tki na chipie)
  * kompatybilne wstecz
* Optymalnej wielkości procesor X razy
  * GPU: 448 rdzeni (w przyszłości 1000ce na chipie)

Stopień zrównoleglania

* rdzeń 10 razy większy jest 2 razy szybszy:
  * Czy lepiej mieć 10 rdzeni małych czy jeden duży?
* Odpowiedź zależy od tego jak duża jest część sekwencyjna.

Prawo Amdahl

P - część równoległa

S - liczba procesorów

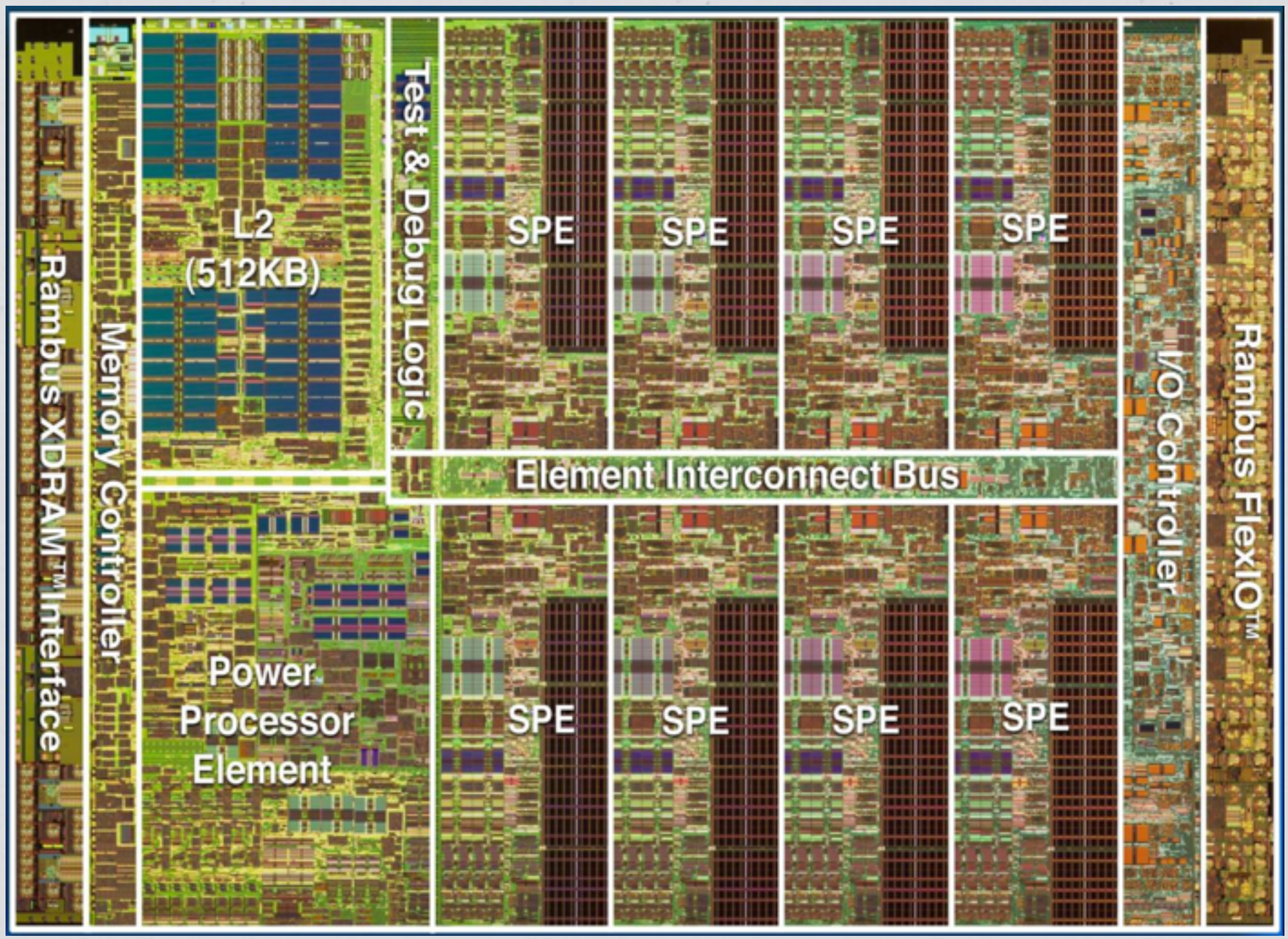
1 / ((1 - P) + P/S) - przyśpieszenie ze zrównoleglania

Dla P = 90% jest to co najwyżej 10 razy.

W praktyce część sekwencyjna 1-P jest mała.

Opłaca się mieć dwa typy procesorów.

Przykład: Cell BE



Przykład: Cell BE

* zaprojektowany przez IBM
* 1 procesor szybki PPE, 8 pomocniczych SPE
* wykorzystany w PlayStation 3
* trudny model programowania:
    * pośredni dostęp do pamięci przez SPE
    * instrukcje wektorowe

Przykład: Cell BE

* miał zastąpić GPU, nie udało się
* mimo zaawansowanego oprogramowania, programiści mieli problemy z wykorzystaniem jego mocy
* IBM wycofał się z rozwoju tej architektury
* procesor Xbox360 nie miał już SPE

PRZYKŁAD GPGPU: NVIDIA CUDA i OpenCL

* OpenCL i CUDA: równoległe funkcje

```
__kernel void fft1D_1024 (__global float2 *in, __global float2 *out,
                          __local float *sMemx, __local float *sMemy) {
  int tid = get_local_id(0);
  int blockIdx = get_group_id(0) * 1024 + tid;
  float2 data[16];

  // starting index of data to/from global memory
  in = in + blockIdx;   out = out + blockIdx;
```

* OpenCL: standard Khronos, szerokie wsparcie sprzętu (CPU, GPU, procesory wektrowe, FPGA)

* często trzeba przesyłać dane po PCI/E (GPU)

Data paralleism

* 2 razy więcej rdzeni => (prawie) 2 razy większa wydajność
* faktyczną liczbę wątków ustala się w trakcie wykonania
  * nie jak dotychczas podczas kompilacji czy konfiguracji
  * może zależeć od dostępnych zasobów
  * w zależności od rozmiaru danych

Przykład: NVIDIA CUDA

CPU:

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

GPU (CUDA):

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Pipeline

* często problem składa się z serii podproblemów
* zamiast zapisywać algorytm sekwencyjnie, można wyrazić je jako przetwarzanie danych
* umożliwia to zazębianie poszczególnych etapów obliczeń

Synchronizacja - obecnie

* z użyciem semafor:
  * wykluczające
  * producent-konsument
* obecnie wykorzystywane, ale są trudne w użyciu
* wysyłanie komunikatów

Synchronizacja - przyszłość

* transakcyjna pamięć
    * zapewnia atomowość całych operacji
* nie ma sprzętu realizującego w pełni ten pomysł, ale koncepcja jest bardzo obiecująca

Przykład: synchronizacja

* N wątków, każdy z kolejką która może pomieścić K zadań
  * wykonuję swoje zadania po kolei, jeśli pojawia się nowe to dokładam je do kolejki
  * co jeżeli kolejka jest pusta/pełna

Przykład: synchronizacja

* task donation: jeśli kolejka pełna to dodajemy zadanie innemu wątkowi do kolejki
* task steal: jeśli kolejka jest pusta, to zabieramy innemu wątkowi je z kolejki
* przydatne w ray-tracing na GPU

Jak skrócić odległość obliczeń od danych?

* lokalność odwołań do pamięci
* RAM w pakiecie z procesorem (procesor wie gdzie jest pamięć)
    * Jest wiele pomysłów, ale każdy ma swoje wady…

Jak skrócić odległość obliczeń od danych?

pomysł	wady
większy cache	synchronizacja, poświęca się na to cenne miejsce
wirtualna hierarchia pamięci	wymaga dodatkowej pracy programisty, podatna na błędy
każdy rdzeń ma swoją lokalną pamięć	trudno opisać programy tak by umiały z tego korzystać

Jak skrócić odległość obliczeń od danych?

pomysł	wady
opis fizyczny	bardzo dużo pracy programisty, nieprzenośne rozwiązanie

PRZYSZŁE ARCHITEKTURY

NVIDIA: Echelon

* projekt Denver: CPU Arm 64 bitowy
* na jednym chip (multiprocesory GPU i procesory ARM)
    * 128 SP + 8 LP -> 20 Teraflop/s
* wirtualne hierarchie pamięci

Intel: Knights Corner

* Larrabee, Knights Ferry, ...
* chip składający się z > 50 rdzeni
* kompatybilny wstecz z x86
* można sterować cache
* operacje na wektorach 512 bitowych

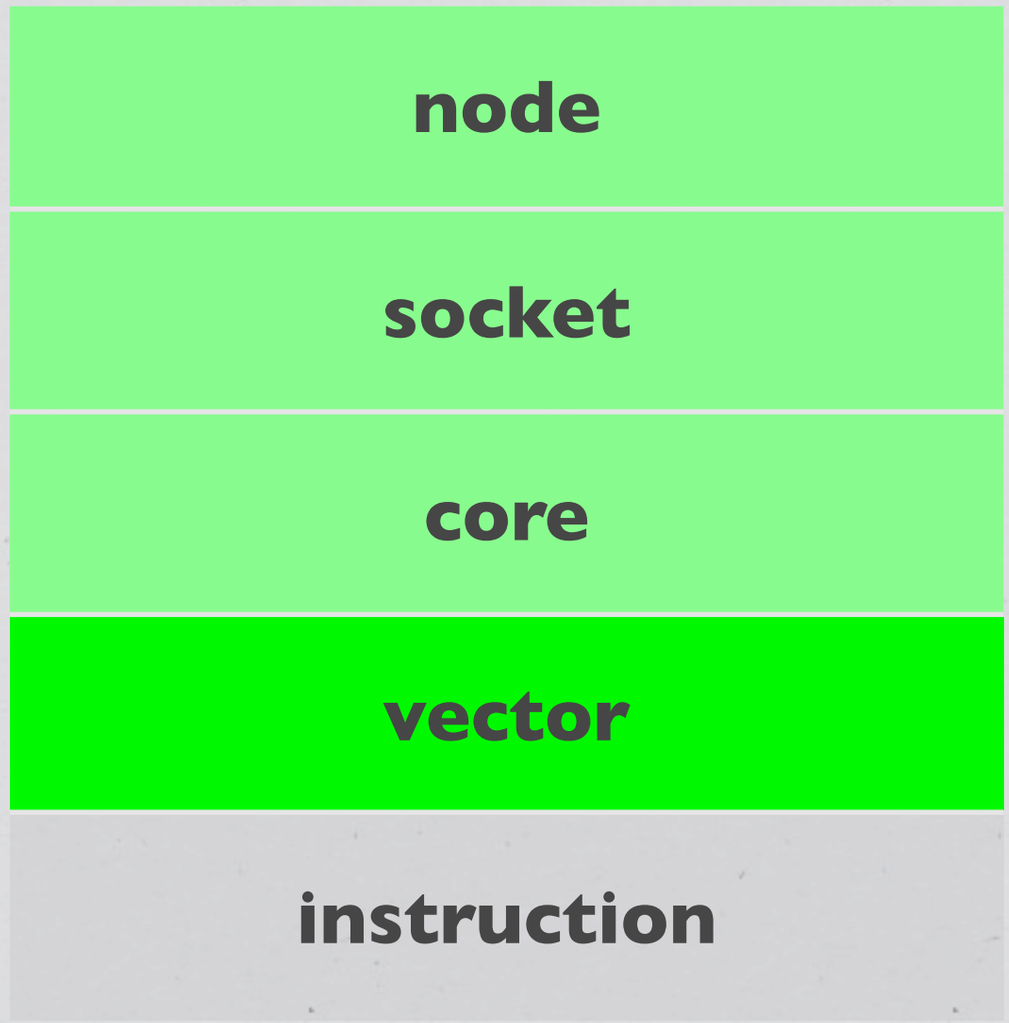
PODSUMOWANIE

Obecne problemy

* Power wall
* Memory wall
* Instruction level parallelism wall

Równoległość

* na wszystkich poziomach



Potrzebny nowy model programowania

* skalowalny ze wzrostem procesorów
* uwzględniający lokalność odwołań do pamięci

PYTANIA?

Inne źródła

* “To Exascale and Beyond” Bill Dally NVIDIA, http://www.nvidia.com/content/PDF/sc_2010/theater/Dally_SC10.pdf

* http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf

Źródła obrazów

* http://domino.research.ibm.com/comm/research_projects.nsf/pages/multicore.CellBE.html

* mapa złóż: http://www.velocitymanager.com/

* http://www.legitreviews.com/article/1484/1/

* kod przykładowy http://en.wikipedia.org/wiki/OpenCL

* http://www.nvidia.com/content/PDF/sc_2010/theater/Dally_SC10.pdf