Introduction to Parallel Computing

Seyong Kim

Department of Physics
Sejong University

Why bother? (used to be)

I want my application to finish quicker!

so I can go home or get published in time for the next review of my grant · · ·

Why bother? (used to be)

G.F. Pfister, *In Search of Clusters, 2nd Ed.*

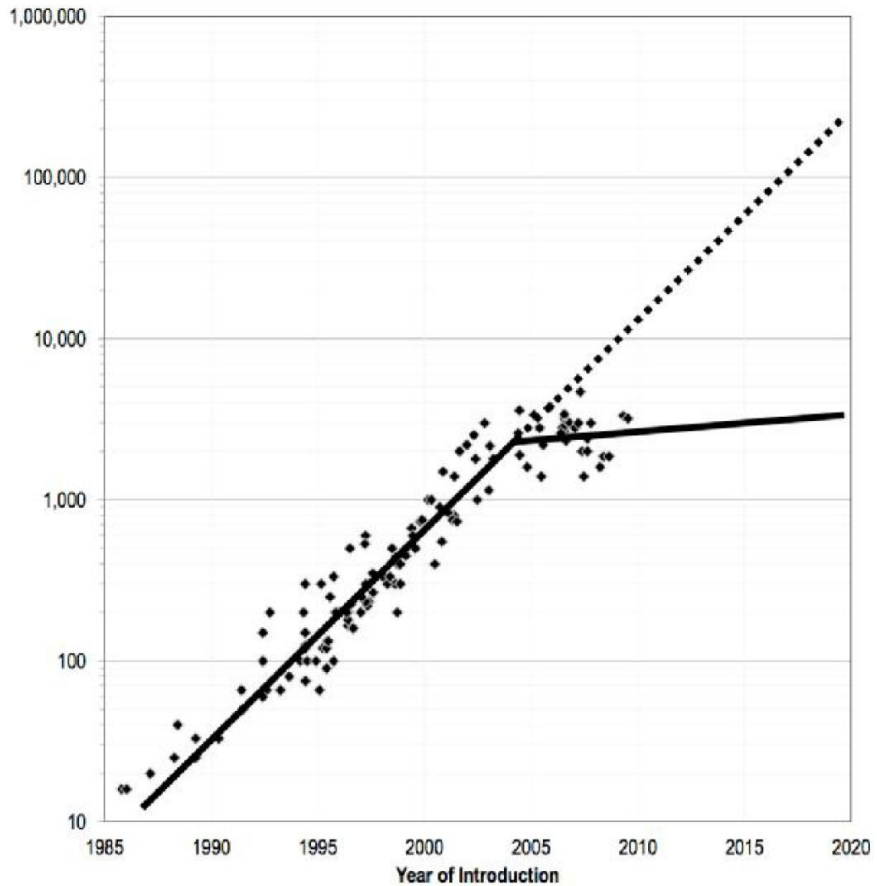
- Work harder
- Work smarter
- Get help

Why bother? (used to be)

Strategy for parallel computing

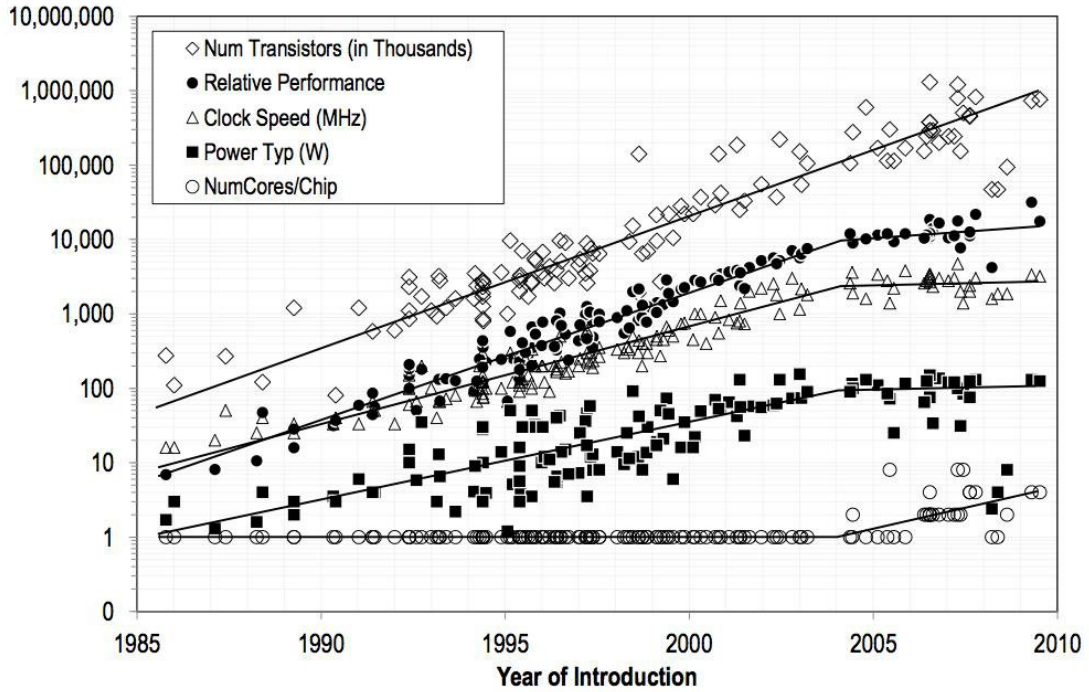
- Don’t do it
- Let someone else do it
- Minimize it

Where we are heading – CPU performance



S.H. Fuller and L.I. Millett, *The Future of Computing Performance: Game Over or Next Level?*

Where we are heading – CPU performance



In S.H. Fuller and L.I. Millett, *The Future of Computing Performance: Game Over or Next Level?*, Data curated by Mark Horowitz with input from: Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç.

Where we are heading – CPU performance

- CMOS chip power consumption: clock speed x voltage^2

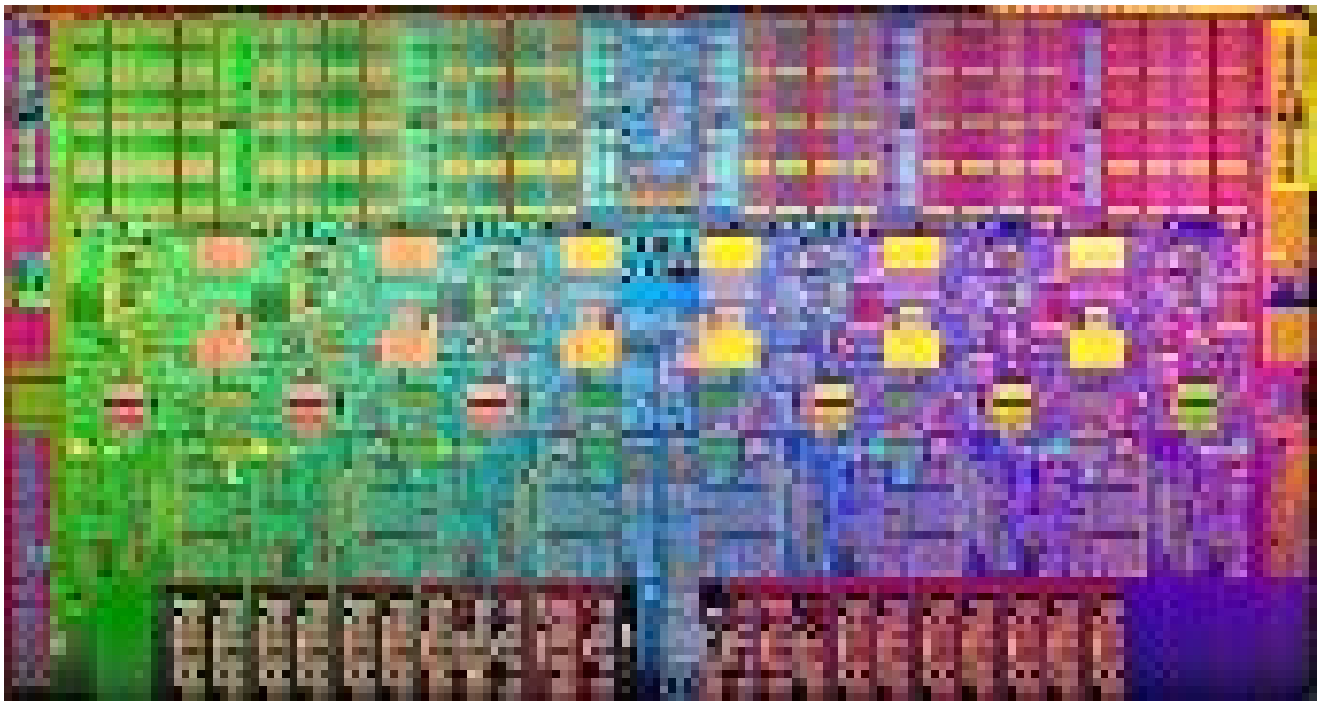
  $\rightarrow$ clock speed is limited

- Moore's law is still valid

  $\rightarrow$ gate density can grow

Where we are heading – CPU performance

- many core path (Intel, AMD $\cdots$)



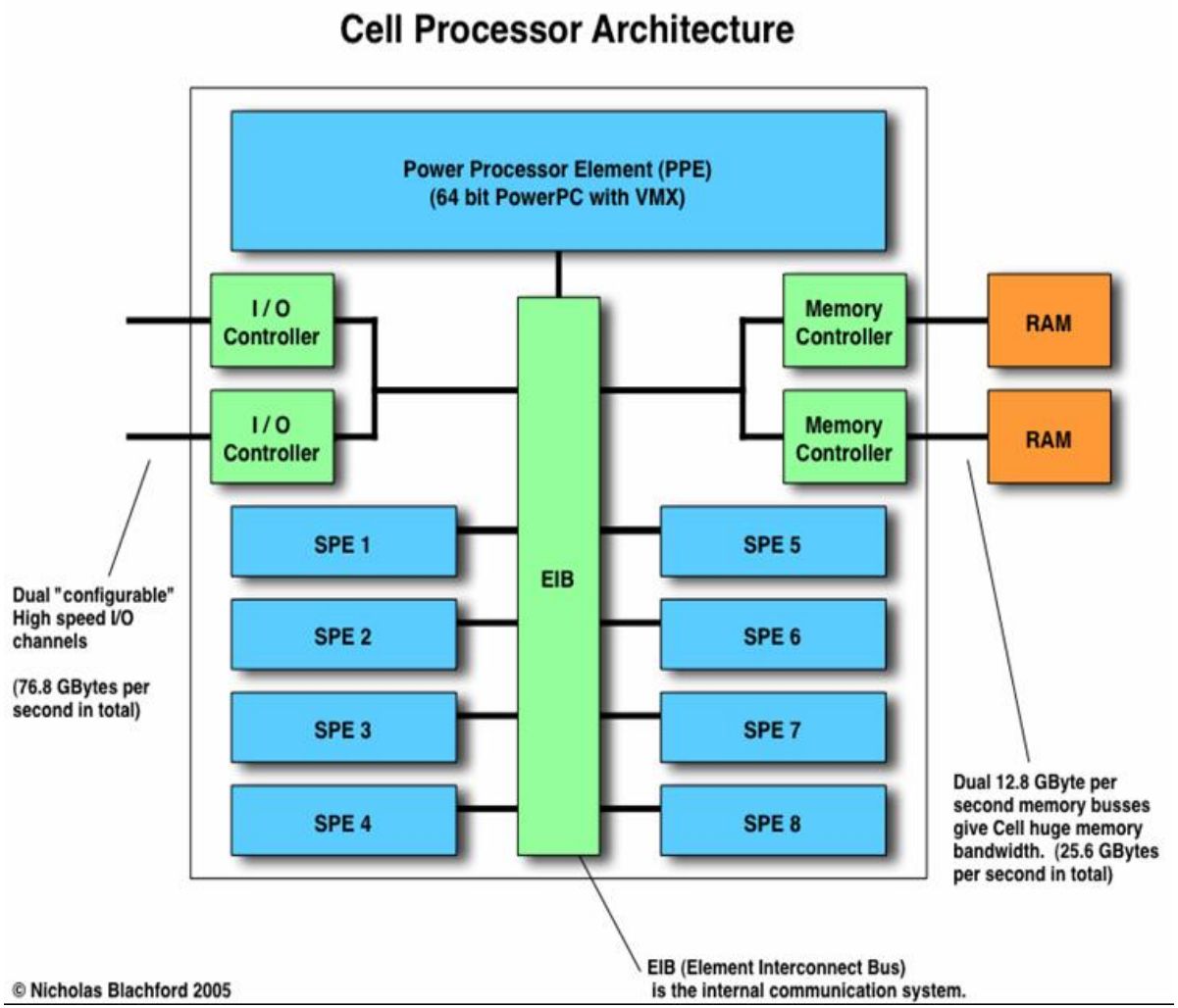
AMD dodeca Core die

Where we are heading – CPU performance

- GPU path (NVIDIA, ATI $\cdots$)

Where we are heading – CPU performance

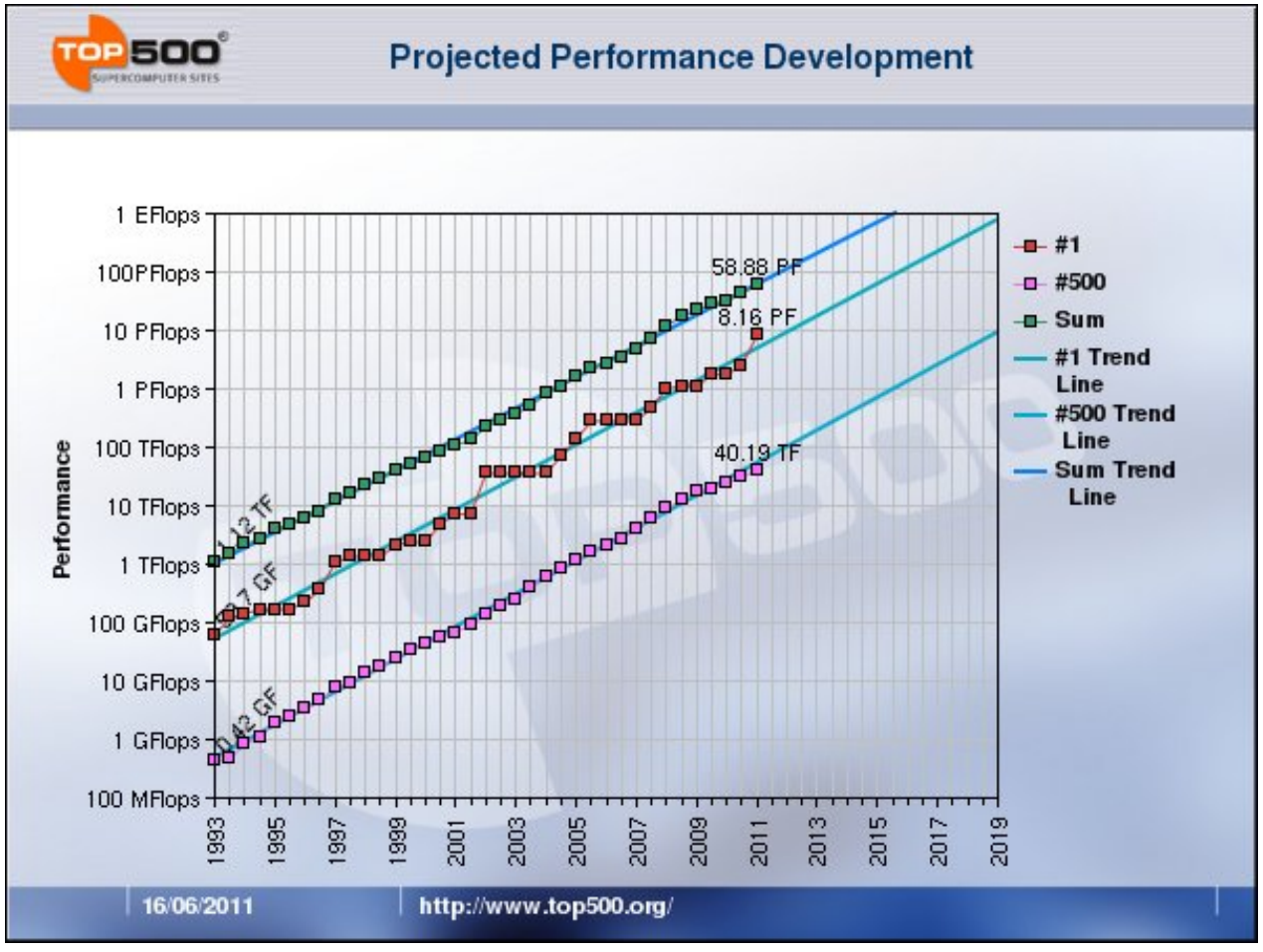
- hybrid (AMD, IBM-Sony-Toshiba, $\cdots$)



Where we are heading – CPU performance

Actually, future CPU architecture has no clear winner yet!

Where we are heading – system architecture



http://www.top500.org

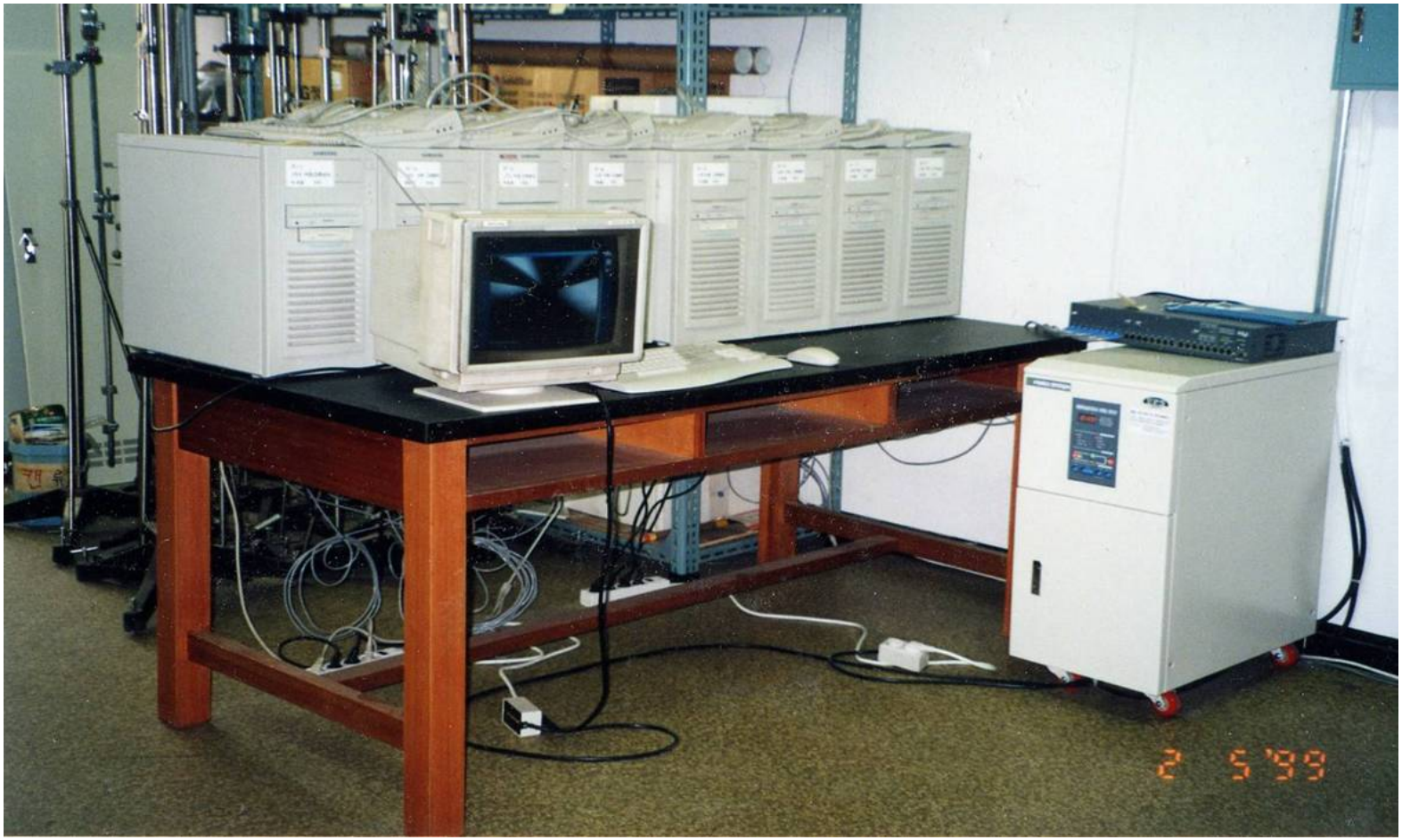
Where we are heading – system architecture

- Korea Meteorological Administration (1999), NEC SX-5, 16 cores
- Korea Meteorological Administration (2004), Cray X1E, 188 cores
- Korea Meteorological Administration (2011), Cray XE6, 45120 cores

Personally – Do It Yourself

1988

Personally – Do It Yourself



1999

Personally – Do It Yourself

2003

Personally – Do It Yourself

Now

Personally – Do It Yourself

near future !

What we mean by parallel computing

one CPU finishes my job in one year

$\rightarrow$ twelve CPU's will finish my job in a month ?

What we mean by parallel computing

So what stops you?

What we mean by parallel computing

What we mean by parallel computing

▲8일 오후 국회 본회의장에서 여야 의원들이 몸싸움을 벌이고 있다(윤창원 기자)

Caveat 1

- some serial algorithm can't be turned into parallel program

```
sum = 0.0;
for(i = 0 ; i < MAX ; i++)
    sum = sum + a[i];
```

Caveat 2

- Amdahl's law

$$\text{speed} - \text{up} = \frac{l_s + l_p}{l_s + \frac{l_p}{N}} \tag{1}$$

for example, $l_s = 0.1, l_p = 0.9$ speed-up = 10 even for $N = \infty$

Caveat 3

- E. Dijstra's concurrency problem

Caveat 4

- Leslie Lamport's sequential consistency and mutual exclusion problem

Everybody starts with x = 1

```
/* in process Able */
x = 1;
x = 2;
x = 3;
...
x = 99;
```

```
/* in process Baker */
y[0] = x;
y[1] = x;
y[2] = x;
...
y[99] = x;
```

Caveat 4

Everybody starts with x = y = 0

```
/* in process Able */
x = 1;
if(y == 0) print (“Able wins!”);
```

```
/* in process Baker */
 y = 1;
if(x == 0) print (“Baker wins!”);
```

In short

- there is no tool which determines that an existing program can be parallelizable

  $\rightarrow$ there exists helping hand (analysis tool)

- if a given algorithm is parallelizable, one needs to determine the best strategy to parallelize

  $\rightarrow$ experiment with domain-decomposition and etc

- affecting the ratio l_p/l_s strongly influences the speed-up of parallelization

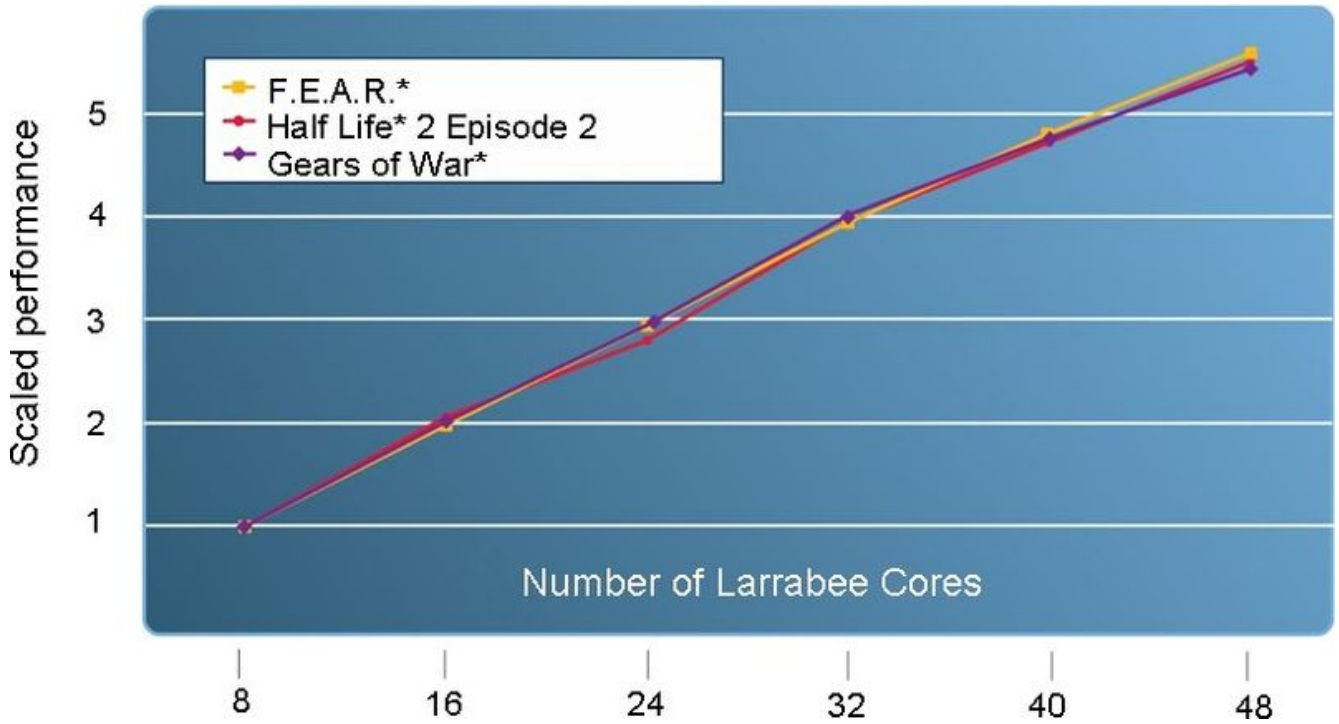
  $\rightarrow$ for example, memory access is part of l_s. experiment with memory access pattern

In short

Even vendor can make a mistake

(cf. Intel's Larabee)

In short



http://en.wikipedia.org/wiki/Larrabee_microarchitecture

Level of parallelization

- fine grained parallelism

  instruction level parallelism: recent GPU architecture

- coarse grained parallelism

  cluster computer

Paradigms of parallel programming

- message passing
    (a) send and received message when needed
    (b) MPI
- data parallel
    (a) operate on parallel data
    (b) OpenMP, multi-thread
- SPMD (Single Program Multiple Data)

How we do parallel computing: an example

- check algorithm for parallelizability
- consider how to balance the load (surface-to-volume ratio)
- the time spent in accessing data is "l_s". devise a way to reduce this overhead
- the result from parallel execution can be different from the result from serial execution due to the order of operation. prepare for debuggin strategy

How we do parallel computing: an example

6iM-^[M-^T 29, 11 20:00 **sum.c** Page 1/1

```
#include <stdio.h>

#define N 1000000

main(int argc,char** argv) {

int   i;
float A[N];
double sum;

for (i = 0;i < N;i++) A[i] = (float)(i+1);

sum = 0.0;

for (i = 0;i < N;i++) sum = sum + A[i];

printf("totalsum = %f\n", sum);

}
```

How we do parallel computing: an example

6ìM-^[M-^T 29, 11 20:15 **reduce_mpi.c** Page 1/1

```
#include <stdio.h>
#include </usr/local/lam-CFC/h/mpi.h>

#define N 1000000

main(int argc, char** argv) {

int   i, node, numtask, subN, iroot;
float A[N];
double sum, sumt;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&node);
MPI_Comm_size(MPI_COMM_WORLD,&numtask);

subN = N/numtask;

for(i = 0;i < subN;i++) A[i] = (float)(i+1 + node*subN);

sum = 0.0;
sumt = 0.0;
iroot = 0;

for(i = 0;i < subN;i++) sum = sum + A[i];

MPI_Reduce(&sum,&sumt,1,MPI_DOUBLE,MPI_SUM,iroot,MPI_COMM_WORLD);

printf("totalsum=%f\n", sumt);

MPI_Finalize();

}
```

How we do parallel computing: an example

- “reduction” example in NVIDIA_CUDA_SDK