

GREEN COMPUTING

W. P. Petersen

Seminar for Applied Mathematics, ETH Zürich

<http://www.math.ethz.ch/~wpp/>

e-mail: wpp@math.ethz.ch

1 November, 2007, Fallstudien

Part-1: what is **green computing**? Let's look at some numbers concerning HPC friendliness to the environment.

- (1) Number one on the **Top500** list is the **Blue Gene/L** machine of **DOE/NNSA/LLNL** (65536 IBM 700 MHz dual-CPU nodes). Electricity requirements are ~ 1.8 mW and cost approximately \$2 million/year. Estimate for the new Jülich **BG/P** machine is 500 kW.
- (2) The **Earth Simulator** in Yokohama (640 **SX-6** nodes) uses 6.4 mW and electricity costs more than \$10 million/year.
- (3) The **NERSC** (Berkeley Lab, 4 big machines) complex's electricity costs are $> \$4$ million/year.
- (4) The new **TACC** 500 Tflop (**AMD** quad cores) takes 2.4 mW plus 1 mW cooling.
- (5) **Gartner Group** predicts by 2011, 70 percent of big US data centers will have disruptions due to energy consumption or costs.



Figure: NNSA BlueGene/L machine



Figure: Oak Ridge Jaguar Cray XT-4 machine



Figure: Yokohama Earth Simulator

Those of us less inflated with our importance should ask: what can we do to get better results for less money/energy?

- ▶ Higher performance but lower energy costs? This seems to mean **multi-core processors**, as we will see in **Part 1**. If we can program them!
- ▶ **Algorithm improvements** have often given us more performance improvements than hardware. See **Part-2**. Humans consume approximately 100 watts (2000 cal/day).

In the following I claim that as a function of frequency ω ,

$$\frac{\text{performance}}{\text{watt}} \sim \omega^{-2}$$

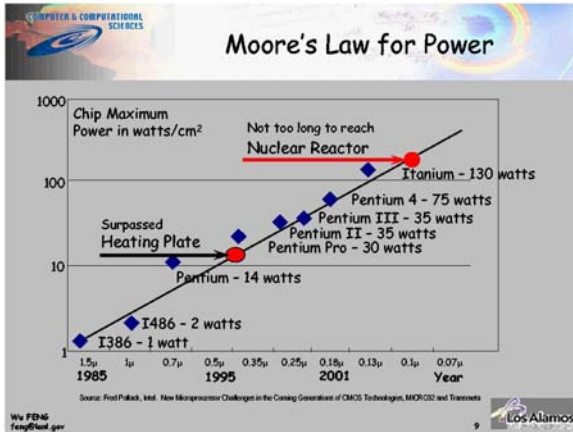


Figure: Microprocessor power density: doubles \sim 2 years. Data from Wu Feng, Los Alamos.

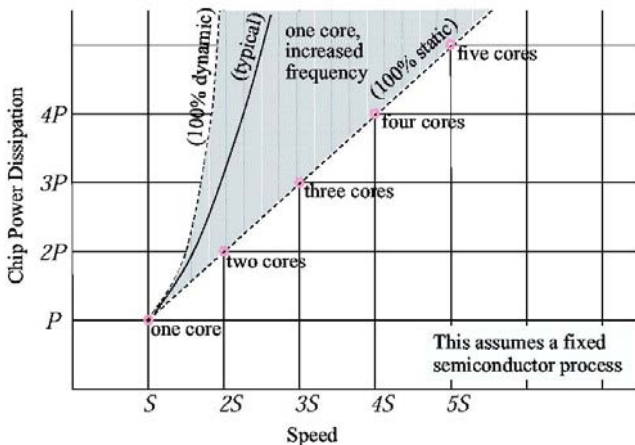


Figure: Power vs. frequency. Source: Burton Smith, Microsoft Corp., keynote address, Int. Supercomputing Conf., June 27, Dresden.

For $P = c \cdot \omega^m$, $m \approx 2$ (typical), $m \approx 3$ (all dynamic).

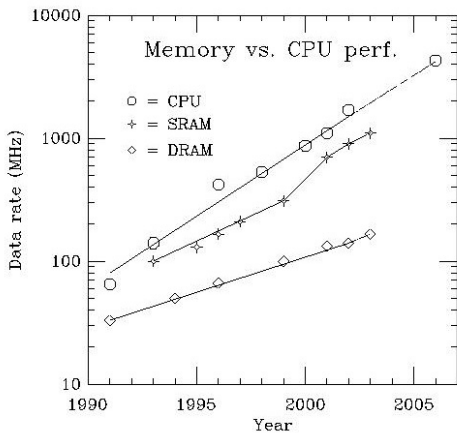


Figure: Memory vs. CPU performance

DRAM is cheap/power efficient, but memory speed \leq 133 MHz.

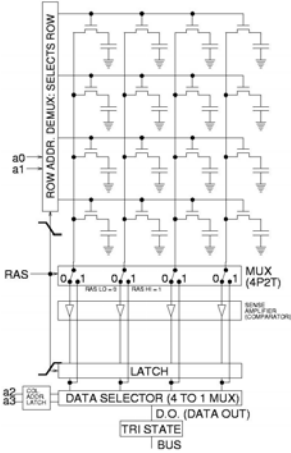


Figure: Dynamic RAM, multiplexed read/write.

What can be done? At fixed frequency (ω), power dissipation looks like

$$P = \frac{1}{2} CV^2$$

so

- ▶ Turn down the voltage, say from 5.2 volts to .8 volts. Below 1 volt, thermal noise starts becoming important.
- ▶ Turn down the frequency.
- ▶ Exploit parallelism.

Important point: hardware, algorithms, and software cannot be decoupled.

Classic benchmark solves linear system for > 1000 unknowns x

$$Ax = b$$

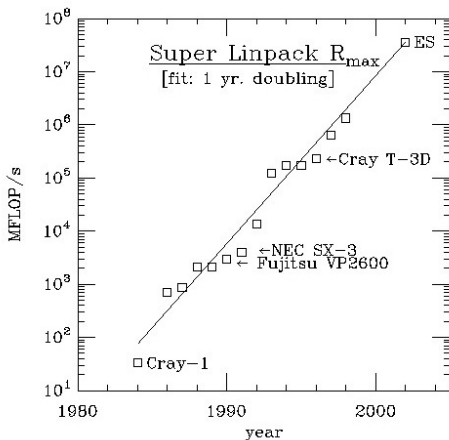


Figure: Super Linpack benchmark results.

Forms of parallelism:

- ▶ pipelines and vectors
- ▶ instruction execution
- ▶ threads on a tight network
- ▶ on a larger network, message passing (MPI, e.g.)

First, **pipelines and vectors**: look at example $\mathbf{C} = \mathbf{A} + \mathbf{B}$

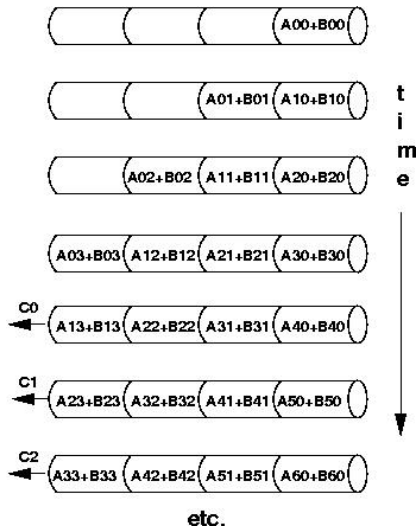


Figure: 4-stage addition pipeline.

It seems paradoxical, but longer latency can increase performance.
The speedup for pipelined operations is

$$\text{speedup} = \frac{Ln}{L + n}$$

where L = latency or pipeline length. If $n \gg L$,

$$\text{speedup} \rightarrow L$$

This is similar to **Little's Law** which says

$$\text{concurrency} = \text{latency} \times \text{bandwidth}$$

For the case that L is larger than the vector length (n),
speedup $\rightarrow n$, which for the IBM CELL, AMD, and Intel vector
hardware is $n = 4$.

- ▶ Pipelines and vectorization on CELL. SPE units each have 128 **vector registers** of four 32-bit words each. **This produces a speedup of 4, the same improvement as using quad-cores.**
- ▶ Automatic vectorization is difficult due to block alignment.
- ▶ *Intrinsics* remain the principal vectorization paradigm: unrolling in groups of 4 (single) or 2 (double) to use these intrinsics.

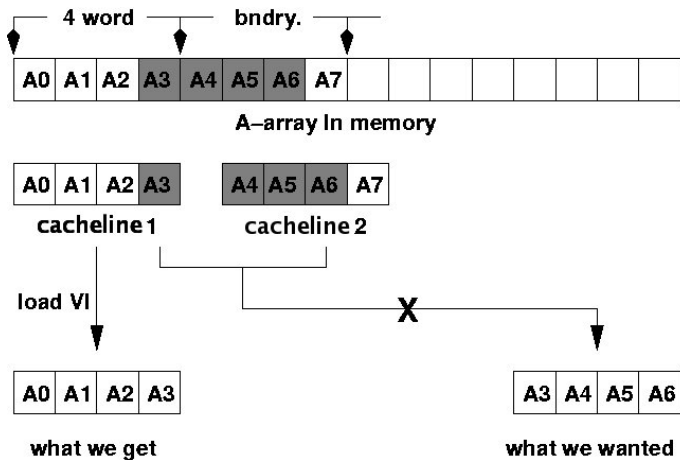


Figure: Short vectors and block alignment.

Levels of parallelism, continued: **instruction level**.

- ▶ Instructions are pipelined. Out-of-order and branch prediction are available only on PPE. SPE executes in linear order with ~ 20 cycle fall-through miss penalty. Typical execution times are 2-7 cycles.

```
if(e(x)){
    y = f(x);
} else {
    y = g(x);
}
```

Multi-core: PS-3 experiments,

- ▶ Price/performance: PS-3 costs \sim \$500/unit with 6-7 working SPEs; cheap compared to say an Opteron 244.
- ▶ more levels of parallelism:
 - * instruction (pipelines) + vectors, CELLS have 128 vector registers.
 - * multiple independent functional units,
 - * independent threads (6-7 SPEs),
 - * message passing between CELLS.
- ▶ Heat: Itanium generates \sim 130 watts, while CELL is only 40-60. Compare 2 Gflops (Itanium) to 45 Gflops (CELL).

The crew:

- ▶ ETH students:

Denis Nordmann, François Gaignat, Stephan Gammeter, Lukas Gamper, Patrick Mächler, Robin Krom, Kaspar Müller, Mauro Calderara, Christine Tobler, Bastian Pentenrieder

- ▶ ETH staff:

Professors: wpp, Rolf Jeltsch, Jörg Waldvogel, Daniel Kressner (math), and Matthias Troyer (Physik).

- ▶ University of Zürich students:

Jonathan Coles, Alessandro Viretta.

- ▶ Univ. of Zürich staff:

Prof. George Lake (CSE/Astrophysik).



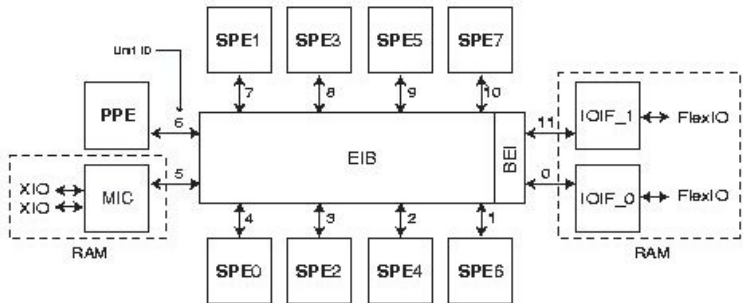
Figure: Informal team from ETH and Univ. ZH. Left to right: Coles, Krom, wpp, Waldvogel, Pentenrieder, Gagnat, Lake, Tobler, Nordmann, and Müller. Our first PS3 is on Gagnat and Lake's knees.



Figure: Matthias Troyer, Lukas Gamper, and Rolf Jeltsch

Current PS-3 machines are poorly suited for scientific/engineering simulations:

- ▶ memories are too small (512 MB with 1/2 of this usable) and no substitution,
- ▶ programming in threads remains painful (Open MP ?),
- ▶ single precision arithmetic is inadequate, double is OK,
 - * 128-bit high precision floating point was written by Jonathan Coles and François Gagnat with help from Mauro Calderara.
- ▶ Giga-bit Ethernet PCI switches are not easily replaced,
- ▶ Open MPI was ported to our 4 PS-3 cluster by Mauro.



BEI	Cell Broadband Engine Interface	MIC	Memory Interface Controller
EIB	Element Interconnect Bus	PPE	PowerPC Processor Element
FlexIO	Rambus FlexIO Bus	RAM	Resource Allocation Management
IOIF	I/O Interface	SPE	Synergistic Processor Element
		XIO	Rambus XDR I/O (XIO) cell

Figure: CELL memory control (CBE programming handbook)

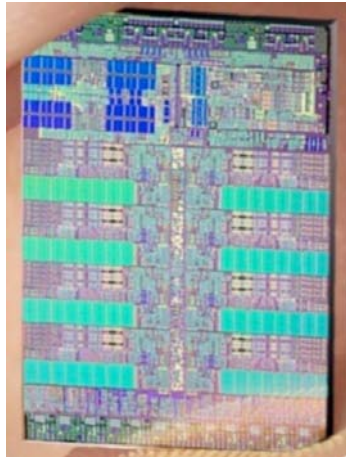


Figure: Basic CELL processor held between thumb and forefinger

Programming so many levels of parallelism is difficult.

- ▶ Vectorization is awkward, usually via intrinsics. Gnu compiler collection, gcc4.2, does some automatic vectorization.
- ▶ SIMD programming on the multi-cores is more awkward: **pthread**s. OpenMP appears to be poorly suited for multi-core due to local DMA access, which means no memory coherency. OpenMP also does too much.
- ▶ **branch prediction** and **out-of-order** execution are energy unfriendly.

We need help from Informatik, particularly compiler gurus, to design/write a set of wrappers or system to provide a **subset** of OpenMP functionality.

An example: compute π by inscribing a unit circle inside a *side* = 2 square. The area of an inscribed quadrant is $\pi/4$.

```
void simulate_on_spe(){
    speid_t spe_ids[SPE_THREADS];
    context ctxs[SPE_THREADS] __attribute__((aligned(16)));
    char* memory[SPE_THREADS];
    int hits=0,i,status;
    for(i=0;i<SPE_THREADS;i++){
        ctxs[i].iterations = iterations/SPE_THREADS;
        ctxs[i].hits =
            (int*)malloc_aligned(16,sizeof(int),memory[i]);
        spe_ids[i] =
            spe_create_thread(0,&bench_spe,&ctxs[i],NULL,-1,0);
        if(spe_ids[i] == 0) exit(1);
    }
}
```

PPE part (cont.)

```
/* Wait for SPE-thread to complete execution.*/
for(i=0; i<SPE_THREADS; i++) {
    (void)spe_wait(spe_ids[i], &status, 0);
}
for(i=0;i<SPE_THREADS;i++) {
    hits += *ctxs[i].hits; free(memory[i]);
}
cout << "Pi = " << 4.0*hits/iterations
}
```

SPE portion

```
#include <math.h> /* Modified IBM/Sony/Toshiba code */
#include <spu_intrinsics.h>
#include <spu_mfcio.h>
#include "../context.h"
volatile context ctx;
int main(unsigned long long spu_id,
         unsigned long long parm)
{
    unsigned int init[16],tag_id = 0;
    int i,hits,InitWELLRNG512a(unsigned int*);
    double sd,ggl(double*);
    spu_writech(MFC_WrTagMask, -1);
    spu_mfcdma32((void *)&ctx,(unsigned int)parm,
                sizeof(context),tag_id,MFC_GET_CMD);
    (void)spu_mfcstat(2); sd = (double) spu_id;
```

SPE portion (cont.)

```
for(i=0;i<16;i++) init[i]=(unsigned int)5.0e+9*ggl(&sd);
InitWELLRNG512a(init);
simulate(ctx.iterations,&hits);
spu_mfcdma32((void *)&hits, (unsigned int)(ctx.hits),
             sizeof(int), tag_id, MFC_PUT_CMD);
(void)spu_mfcstat(2); return (0);
}
```

Compare to: OpenMP,

```
#pragma omp for
#pragma omp shared(n,thits) private(hits)
for(k=0;k<NTREADS;k++){
    simulate(n,hits)
    thits += *hits;
}
```

Actual calculation:

```
void simulate(int n, int* hits){
    int i,hits_count=0;
    double WELLRNG512a();
    double x,y;
    for(i=0;i<n;i++){
        x = WELLRNG512a(); y = WELLRNG512a();
        if ((x*x+y*y) < 1.0) hits_count++;
    }
    *hits=hits_count;
    return;
}
```

Compilers are ppuxlc++ amd spuxlc

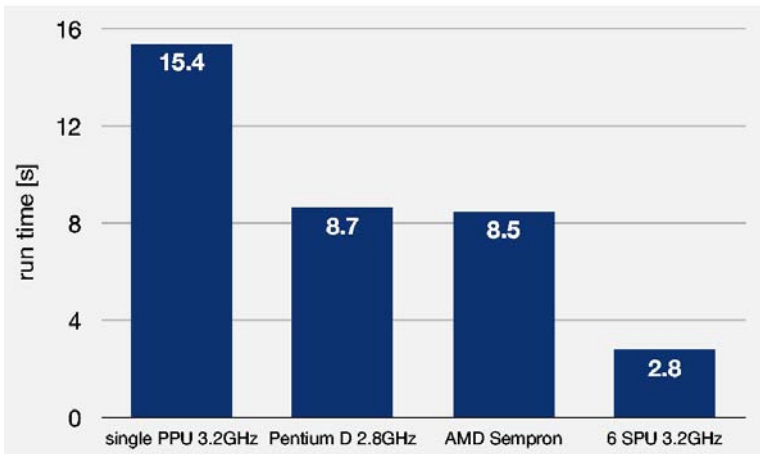


Figure: Six SPE results: **No vectorization**. Stephan Gammeter data using IBM code and srand48.

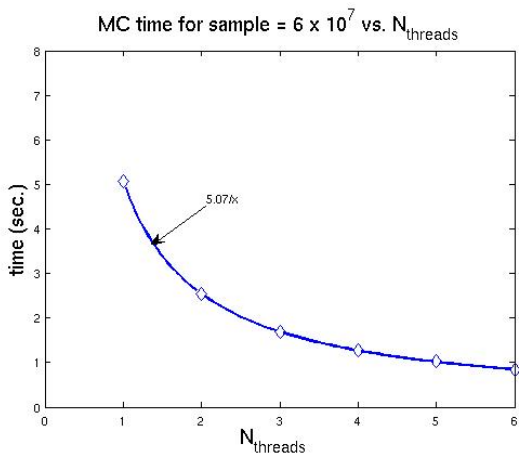


Figure: MC comp. time for π with $N_{\text{sample}} = 6 \times 10^7$ vs. N_{threads} . Speedup=5.9. wpp data using WELLRNG512a.

Another example: self-sorting FFT

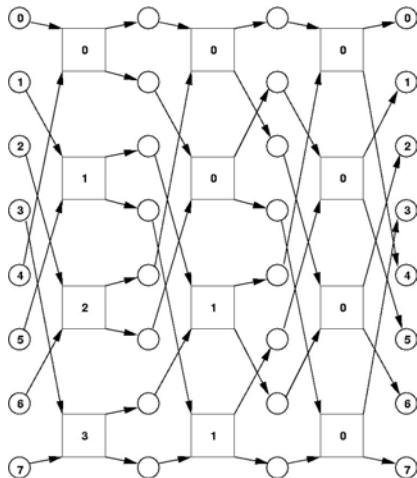


Figure: Workspace version of self-sorting FFT

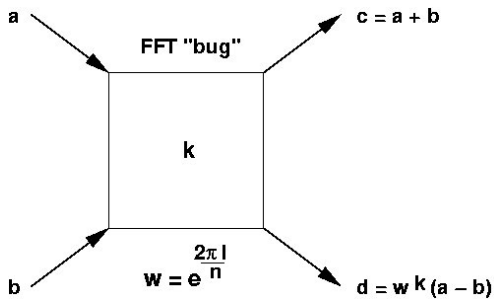


Figure: Decimation in time computational FFT "bug"

$$V_3 = \begin{bmatrix} (a-b)_r \\ (a-b)_i \\ (a-b)_r \\ (a-b)_i \end{bmatrix}, \quad V_6 = \begin{bmatrix} \omega_r \\ \omega_r \\ \omega_r \\ \omega_r \end{bmatrix}, \quad V_7 = \begin{bmatrix} -\omega_i \\ \omega_i \\ -\omega_i \\ \omega_i \end{bmatrix},$$

$$V_3 \xrightarrow{\text{shuffle}} V_4 = \begin{bmatrix} (a-b)_i \\ (a-b)_r \\ (a-b)_i \\ (a-b)_r \end{bmatrix}, \quad V_0 = V_6 \cdot V_3 = \begin{bmatrix} \omega_r(a-b)_r \\ \omega_r(a-b)_i \\ \omega_r(a-b)_r \\ \omega_r(a-b)_i \end{bmatrix},$$

$$V_1 = V_7 \cdot V_4 = \begin{bmatrix} -\omega_i(a-b)_i \\ \omega_i(a-b)_r \\ -\omega_i(a-b)_i \\ \omega_i(a-b)_r \end{bmatrix}, \quad \mathbf{d} = V_2 = V_0 + V_1. \quad (\text{result})$$

Figure: Complex arithmetic for $\mathbf{d} = w^k(\mathbf{a} - \mathbf{b})$ using Altivec. $\mathbf{c} = \mathbf{a} + \mathbf{b}$ is easy.

Workspace version of FFT (used ppxl1c -03 -qaltivec, and alignment via valloc):

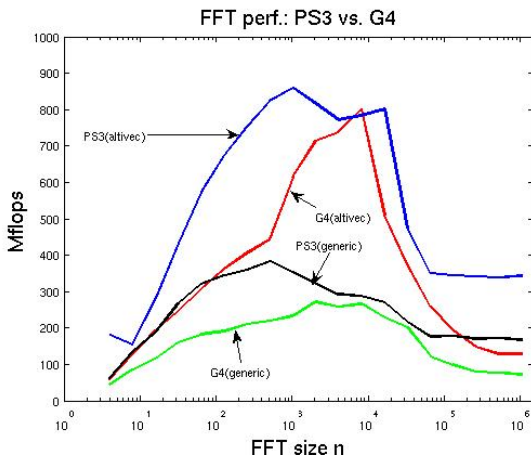


Figure: PS-3 PPE vs. G-4 (Ogdoad), generic vs. AltiVec.

With much loop unrolling and twiddle-factor expansion.

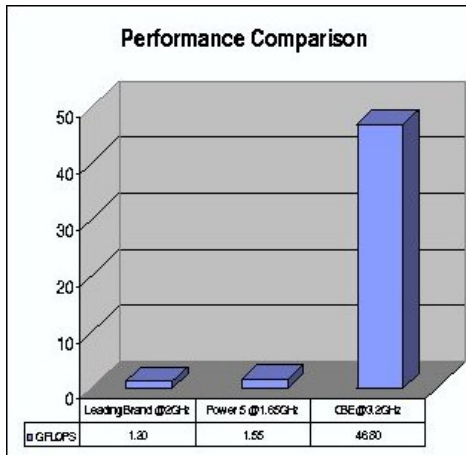


Figure: Chunghen Chow et al, Large (2^{24}) FFT report data.

Multiple PS3 experiments: PCI switches are for Giga-bit Ethernet, but unchangeable

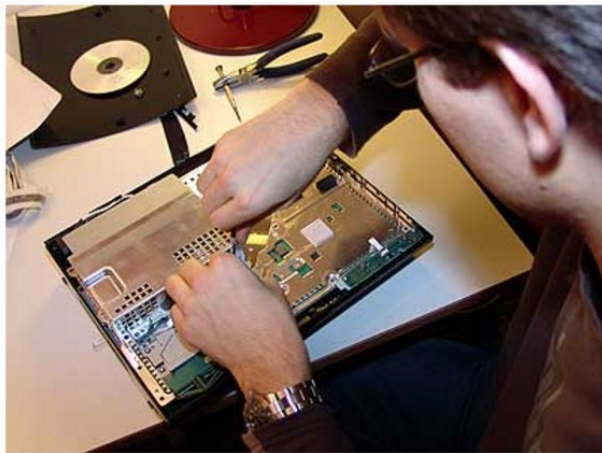


Figure: Inside a PS3: roundtrip message passing $\geq 220\mu\text{S}$

Part-2: the importance of algorithms. Thinking is green: thinking about hardware designs and thinking about algorithms to be able to use the hardware efficiently.

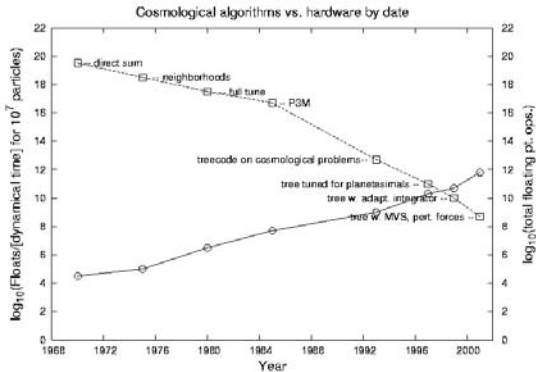


Figure: Comparing hardware vs. algorithms: G. Lake data.

Algorithms for N -body simulations. For N particles interacting gravitationally, the force on particle i is

$$\mathbf{F}_i = G \sum_{j \neq i} \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|^3} = \sum_{j \neq i} \mathbf{f}_{i,j}$$

which seems to require $N \cdot (N - 1)/2$ evaluations. Fortunately, this is not entirely necessary. The contributions may be split into two parts,

$$\mathbf{F} = w\mathbf{F}^{SR} + (1 - w)\mathbf{F}^{LR}$$

For example, in P^3M codes, this splitting is

$$\mathbf{F}_i^{SR} = \sum_{j \in \text{neighbor}(i)} \mathbf{f}_{i,j}$$

$$\mathbf{F}_i^{LR} = -\nabla_i \Phi(\mathbf{x}_1, \mathbf{x}_2, \dots)$$

- ▶ Potential $\Phi(\mathbf{x}_1, \dots)$ computed by convolving a Green function with the density $\rho(\mathbf{x})$, using FFT.
- ▶ Density $\rho(\mathbf{x})$, Charge assignments use TSC (triangular shaped cloud), e.g. Charge q_K assigned to cell K with **center** \mathbf{x}_K For each particle, with coordinate \mathbf{x}_j , its density function is $\rho_j(\mathbf{x}) = m_j \delta(\mathbf{x} - \mathbf{x}_j)$. The charge assignment to cell K is

$$q_K = \sum_j S_K * \rho_j = \int S_K(\mathbf{x} - \mathbf{y}) \sum_j \rho_j(\mathbf{y}) d\mathbf{y}$$

where $S_K(\mathbf{x})$ is a shape functions for cell K , e.g. TSC:

$$S_K(\mathbf{x}) = 0 \vee (1 - |\mathbf{x} - \mathbf{x}_K|)$$

Ewald's weighting is given by $erfc(\alpha|\mathbf{x} - \mathbf{x}_K|)$, with $0 < \alpha < 1$, a tuning parameter. Usually, $\alpha = 2/L$, where the number of cells is L^3 . Here is what $erfc$ looks like

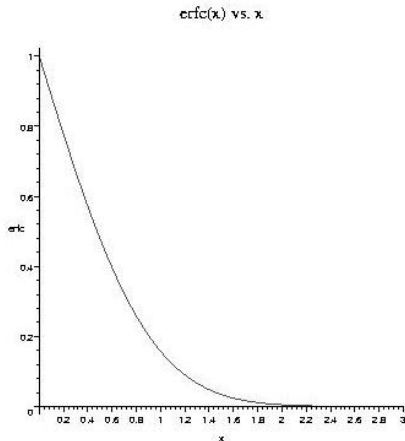


Figure: Complementary error function

- ▶ With adaptive refinement, and $N \approx L^3$, the AP^3M algorithm scales approximately as $O(N \log N)$ operations/time step.
- ▶ When the system is extremely clustered, it can be $O(N^2)$ without unlimited refinement.

So, AP^3M works well until clusters dominate. Then **trees** work better.

Tree code idea: a group of particles $\{j\}$ at distance r from i , is **seen** to have an angular diameter α , that is

$$\alpha = \frac{\max |\mathbf{x}_j - \mathbf{x}_{j'}|}{r}.$$

If α is small enough, say $\alpha < \theta$, the whole group of j -s may be treated as a single particle with mass $\sum_j m_j$ at distance r .

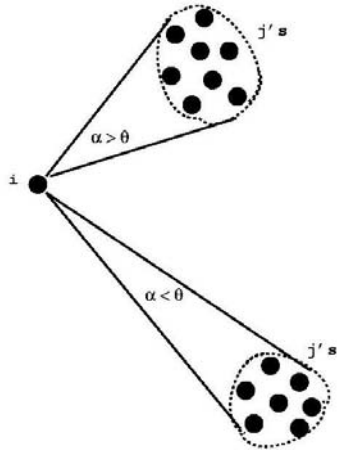


Figure: Opening angle acceptance criteria

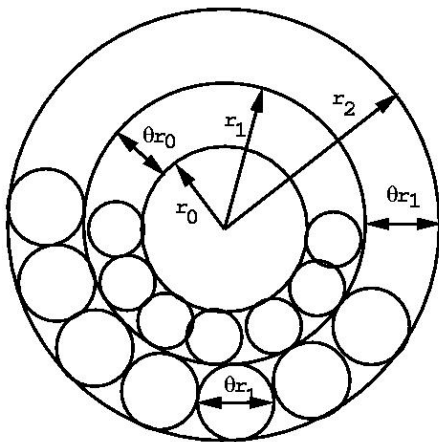


Figure: Hernquist's argument about operation counts for trees

Operation count for trees. This is Lars Hernquist's argument. Let θ be an acceptance angle criteria. If a group appears to particle i as size s , and

$$\frac{s}{r} \leq \theta$$

then the group can be treated as a simple lump at distance r . $\theta \sim 1/2$ is typical. The mean number density n is

$$n = \frac{N}{\text{volume}} = \frac{N}{\frac{4\pi R^3}{3}}$$

The number of interactions between a particle at the center and the **lumps** in shells at radii r_i is

$$n_{int}(\text{per shell}) = \sum_i n_i$$

where the number of **lumps** at r_i is

$$\frac{\text{volume of shell}}{\text{volume of lump}}.$$

Or, where n_{sh} is the number of shells,

$$n_i \sim \frac{(4\pi r_i^2)(\theta r_i)}{\frac{4\pi}{3}(\frac{\theta r_i}{2})^3} = \frac{24}{\theta^2}$$
$$n_{int} = \frac{24}{\theta^2} n_{sh}$$

Now, the inner-most sphere has only direct particle-particle interactions, where

$$s = \frac{1}{n^{1/3}}$$

thus has radius

$$r_0 = \frac{1}{\theta n^{1/3}}$$

and the number of particles there is

$$n_0 = N\left(\frac{r_0}{R}\right)^3 = \frac{4\pi}{3\theta^3}.$$

Successive radii are at

$$r_i = r_0(1 + \theta)^i$$

so

$$n_{sh} = \frac{\log R/r_0}{\log(1 + \theta)} = \frac{\log \theta (\frac{3N}{4\pi})^{1/3}}{\log(1 + \theta)}$$

So altogether,

$$\begin{aligned} n_{int} &= n_0 + \frac{24}{\theta^2} n_{sh} \\ &= \frac{4\pi}{\theta^3} + \frac{24}{\theta^2} \frac{\log \theta (\frac{3N}{4\pi})^{1/3}}{\log(1 + \theta)} \\ &\sim (\text{const.}) \frac{\log N}{\theta^2} \end{aligned}$$

per particle, or over all particles

$$(\text{const.}) N \frac{\log N}{\theta^2} = O(N \log N).$$

Improvements:

- ▶ If $\alpha \sim \theta$, compute force contribution by expanding \mathbf{F} in multipoles. Stadel's kd -tree does this to hexadecapole order. †
- ▶ If particles are relatively remote, they may be moving fast but see small forces. Thus, do not compute these forces every step but use old force and velocity data for these. ‡
- ▶ To conserve the energy, use symplectic, or near symplectic integrators. †‡.

† Joachim Stadel's talk will be in the Fallstudien on 13.12.07.

‡ George Lake's talk in the Fallstudien was on 16.01.06

Another example, **Monte-Carlo**

- ▶ MC is usually embarrassingly parallel - sample paths are independent.
- ▶ Independent random number streams are easily arranged. Typical RNGs take two forms:

$$x_n = x_{n-p} \text{ op } x_{n-q}, \text{ or}$$
$$x_n^{[i]} = x_{n-p}^{[i]} \text{ op } x_{n-q}^{[i]}$$

Where the seed buffer is larger than $p \vee q$. The stream version has $i = 1 \dots N_{streams}$.

L'Ecuyer and Panneton's WELLRNG512a uses a circular buffer containing sixteen integers.

There has been considerable progress in MC methods for statistical physics.

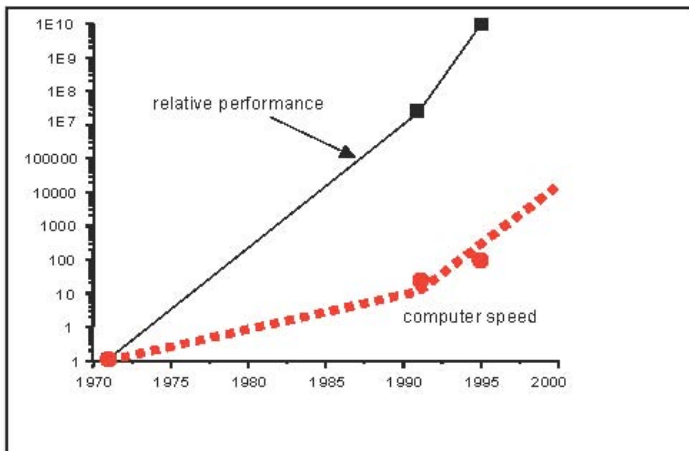


Figure: Algorithms vs. hardware. Data from David Landau. †

†David Landau's talk in the Fallstudien was on 2 June, 2005.

Field theory algorithms, e.g. lattice (or graph (V, E)) of spins

$$\mathcal{H} = \mathcal{H}(\mathbf{x}), \quad \text{e.g.} \quad \mathcal{H} = \sum_{\langle i,j \rangle \in E} 1[x_i \neq x_j]$$

where usual problem ($\beta = 1/kT$) is to determine the partition function

$$\mathcal{Z} = \text{tr} e^{-\beta\mathcal{H}}.$$

and its functionals

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \text{tr} \left[\mathcal{O}(\mathbf{x}) e^{-\beta\mathcal{H}} \right]$$

Usual algorithms equilibrate \mathbf{x} such that Markov chain

$$\dots \mathbf{x}^{[i]} \rightarrow \mathbf{x}^{[i+1]} \rightarrow \mathbf{x}^{[i+2]} \dots$$

samples the phase space. **Detailed balance** is preserved:

$$\frac{p(\mathbf{x}^{[i]} \rightarrow \mathbf{x}^{[i+1]})}{p(\mathbf{x}^{[i+1]} \rightarrow \mathbf{x}^{[i]})} = \frac{e^{-\beta\mathcal{H}(\mathbf{x}^{[i+1]})}}{e^{-\beta\mathcal{H}(\mathbf{x}^{[i]})}}$$

steps $\mathbf{x}^{[i]} \rightarrow \mathbf{x}^{[i+1]}$ accepted with probabilities determined by the algorithm - say heat bath or Metropolis.

- ▶ Heat bath (Gibb's sampler)

$$p(\mathbf{x}^{[i]} \rightarrow \mathbf{x}^{[i+1]}) = \frac{e^{-\beta\mathcal{H}(\mathbf{x}^{[i+1]})}}{e^{-\beta\mathcal{H}(\mathbf{x}^{[i]})} + e^{-\beta\mathcal{H}(\mathbf{x}^{[i+1]})}}$$

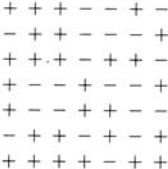
- ▶ or Metropolis

$$p(\mathbf{x}^{[i]} \rightarrow \mathbf{x}^{[i+1]}) = \min\left(1, \frac{e^{-\beta\mathcal{H}(\mathbf{x}^{[i+1]})}}{e^{-\beta\mathcal{H}(\mathbf{x}^{[i]})}}\right)$$

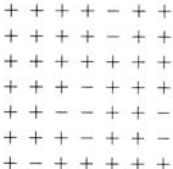
Improvements, for graph $G = (V, E)$,

- ▶ Blocking algorithms, which assign collective spins to clusters. For example, Swendsen-Wang. Not all samples are available: $\sim 2^{|V|}$, so clustering is a significant improvement.
- ▶ De-correlation methods to reduce artificial correlations caused by the algorithm's updates.
- ▶ Weighting or importance sampling methods - for example weighting the clusters.
- ▶ Parallelization.

Correlations for small vs. large values of β



(a)



(b)



(c)

Figure: When β is small, system is disordered; when β is large, it is much more ordered.

Swendsen-Wang groups vertices into clusters, with net *spin*, $\langle \mathbf{x} \rangle_{cluster}$. Instead of $p(\mathbf{x}) = e^{-\beta \mathcal{H}(\mathbf{x})} / \mathcal{Z}$, auxiliary variables $\mathbf{Z} \in \{0, 1\}^{|E|}$ and a joint $p(\mathbf{x} | \mathbf{Z})$ are used. The marginal of this distribution is $p(\mathbf{x})$.

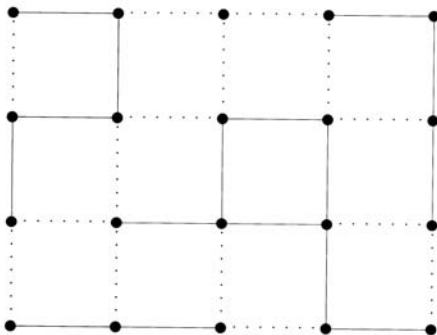


Figure: There are nine clusters here, including singlet

Conclusions and outlook:

- ▶ Scaling of performance/watt is worse than $O(\omega^{-1})$ and probably closer to $O(\omega^{-2})$.
- ▶ Pushing down the voltage is helpful, but error rates go up.
- ▶ Parallelism helps performance/watt. Programming multi-core is painful.
- ▶ Algorithm improvements give better results, but cannot be decoupled from hardware, nor from programming paradigm.
- ▶ We need Informatik support to make programming easier.
- ▶ THINKING IS GREEN.

References:

- ▶ <http://cluster.qubits.ch/>
- ▶ François Gaignat, **Multiplication of hfn numbers**, project report, Oct. 2007.
- ▶ A. Chow, G.C. Fossum, and D.A. Brokenshire, **A Programming Example: Large FFT on the Cell Broadband Engine**, IBM Technical Report, 2005.
- ▶ http://www-128.ibm.com/developerworks/power/cell/docs_articles.html
- ▶ Neal Madras, *Lectures on M-C Methods*, AMS/Fields Inst., 2002.
- ▶ S. Pfalzner and P. Gibbon, *Many-Body Tree Methods in Physics*, Cambridge Univ. Press, 1996.
- ▶ M. Feldman, **Are We Green Yet?**, HPC-Wire, 18.10.07, vol. 16, no. 42. Also: http://en.wikipedia.org/wiki/Green_computing