HPC Overview

LCI 2012
École d'été sur les Clusters de calcul HPC

Ronan KERYELL (Ronan.Keryell@silkan.com)
Chief Scientist

SILKAN / Wild Systems

5201 Great America Parkway #320
Santa Clara, CA 95054, USA

SILKAN (formerly HPC Project)
Cap Omega, Rond-point Benjamin Franklin, CS 39521
34960 Montpellier Cedex 2, France

07/04/2012

Outline

Outline

World modeling

- Old dream of humanity: better understanding the world
- $\leadsto$ Mathematics development for understanding
- Prevision through mathematics formalism
- Complex systems $\leadsto$ no analytic solution (often)
- Huge slow computations, impacted by human errors

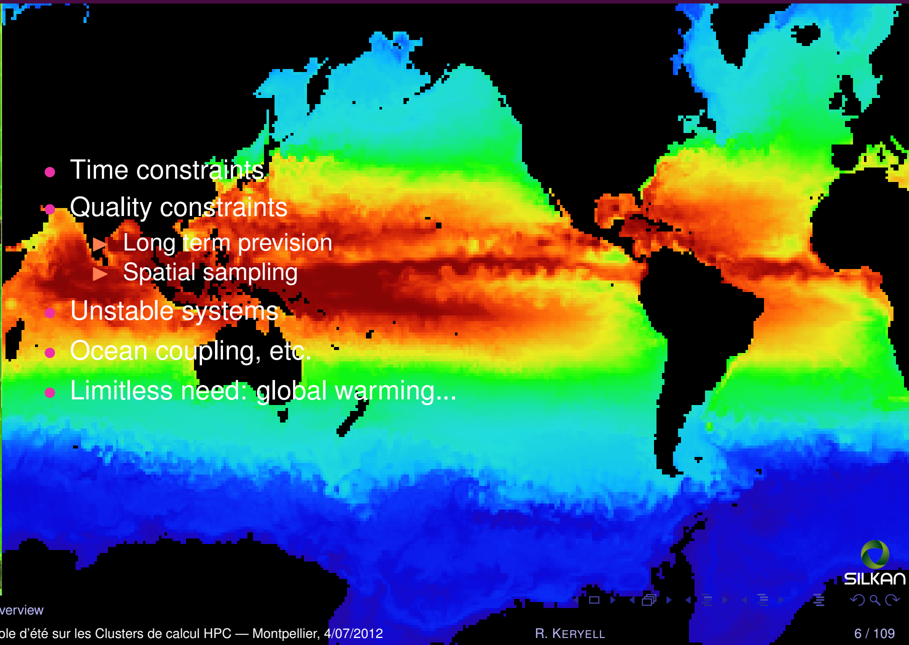
Development of computers for:

- Bigger models : more precise previsions
- Task automation (finance, management)
- Getting results quicker

Some applications

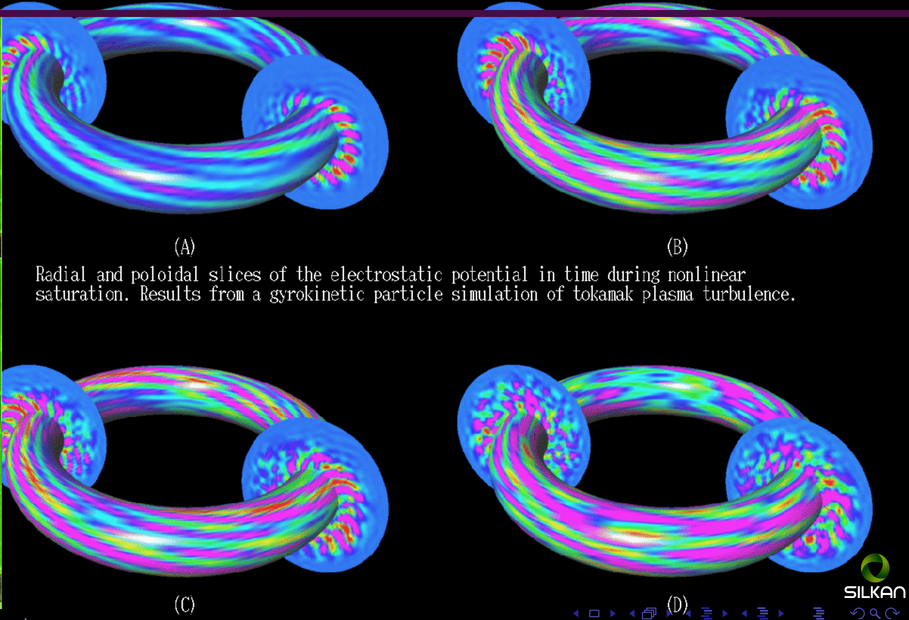
- Simulation and scientific computing, experimental numerical design
  - Environnement
  - Molecular biology
  - Industrial design (cars, golf...)
  - *Ab initio* chemistry
  - Military simulations (US ASCI Accelerated Strategic Computing Initiative...)
- Big data (bases) & *data-mining*, social networks, WWW computing...
- Finance simulations, option pricing...

Meteorology



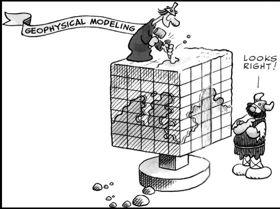
- Time constraints
- Quality constraints
  - Long term prevision
  - Spatial sampling
- Unstable systems
- Ocean coupling, etc.
- Limitless need: global warming...

Tokamak — nuclear fusion



Constraints on computers

- More pieces of information to store (finer sampling) ↗ memory



- More computations to do, ↗ computation speed
  - Throughput : many computations/time units
  - Latency : execution time of one task
- Applications limited by performance
- Needs: always more!

  μηδὲν ἄγαν
- Measures with benchmark programs

Fastest computers at some time (1–4 order of magnitude): *supercomputers*

But big improvements on algorithms too...

Outline

SILKAN

Top 500

http://top500.org

- List 500 biggest (registered) computers in the world since 1993
- Top 10 : even more select
- LINPACK Benchmark : *LU* matrix factoring
  - More $\mathcal{O}(n^3)$ computing than $\mathcal{O}(n^2)$ communications
  - Regular test case not often encountered in real life
  - To be considered as a peak performance

Estimation of future “standard” computing technology development

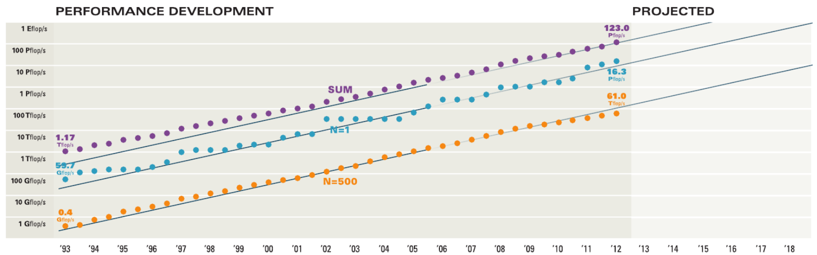
Top 10 — June 2012

TFLOPS performance

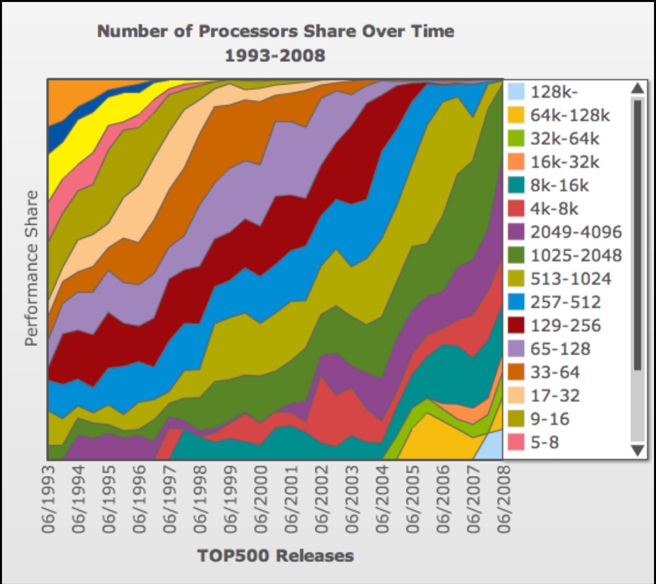
Rank	Site	Computer/Year Vendor	Cores	Rmax	Rpeak	Power (kW)
1	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom / 2011 IBM	1 572 864	16 325	20 133	7 890
2	RIKEN Advanced Institute for Computational Science (AICS), Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705 024	10510	11280	12660
3	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	786432	8162	10066	3945
4	Leibniz Rechenzentrum Germany	SuperMUC - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR / 2012 IBM	147456	2897	3185	3422
5	National Supercomputing Center in Tianjin, China	Tianhe-1A - NUDT YH MPP, X5670 2.93 Ghz 6C, NVIDIA C2050 / 2010 (NUDT)	186 368	2566	4701	4040
6	DOE/SC/Oak Ridge National Laboratory, USA	Jaguar - Cray XK6, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA 2090 / 2009 Cray Inc	298592	1941	2628	5142
7	CINECA Italy	Fermi - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	163840	1725	2097	822
8	Forschungszentrum Juelich (FZJ) Germany	JuQUEEN - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	131072	1380	1678	657
9	CEA/TGCC-GENCI France	Curie thin nodes - Bullx B510, Xeon E5-2680 8C 2.700GHz, Infiniband QDR / 2012 Bull	77184	1359	1667	2251
10	National Supercomputing Centre in Shenzhen (NSCS), China	Nebulae - Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120 640	1271	2984	2580

http://www.top500.org/list/2012/06/100

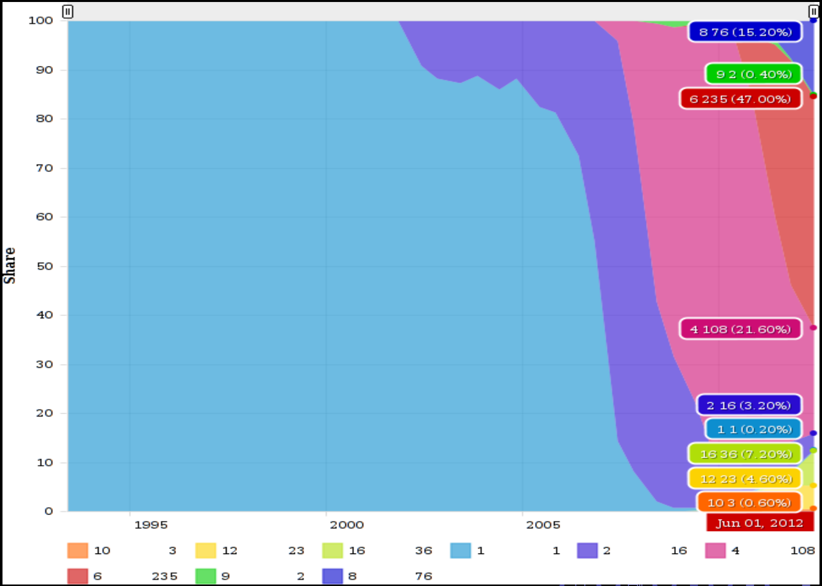
Total performance



Toward massive parallelism — 06/2008



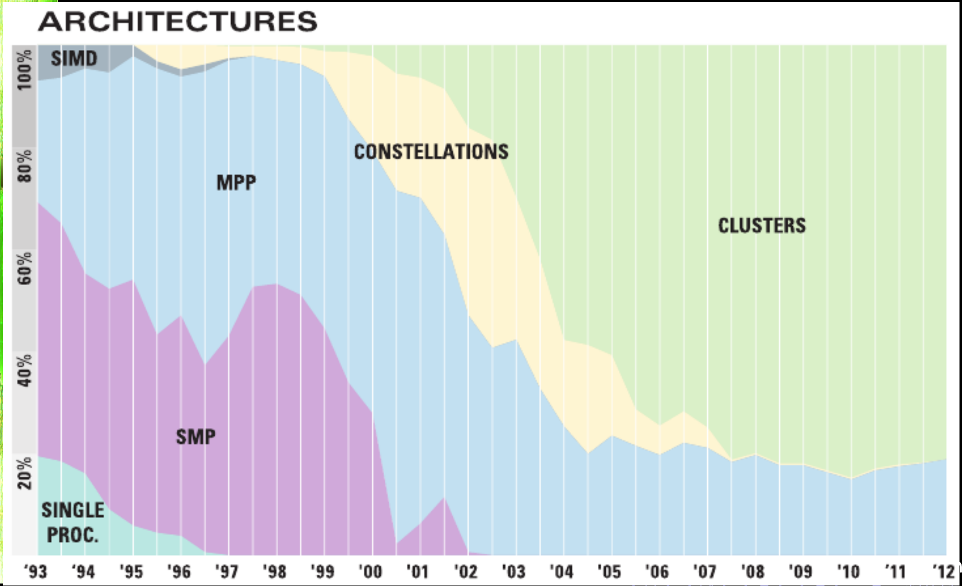
Number of cores per chip



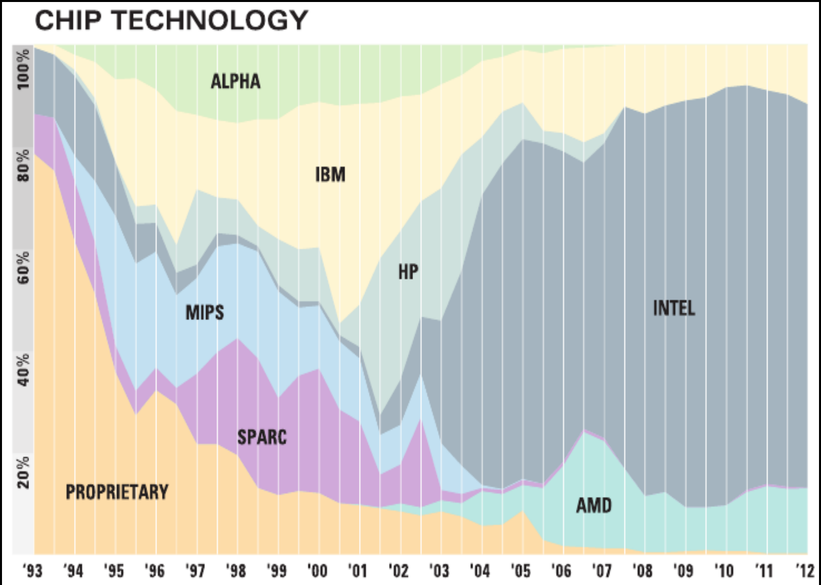
Types of parallel machine architectures *(I)*

- Vector: processors optimized to compute with vectors
- Clusters: “standard” computers interconnected with networks
- MPP (*Massively Parallel Processors*): more specialized computers with specific networks
- SMP (*Shared Memory Processor*) : processors sharing common memory (any commodity multicore computer today)
- Constellations : clusters of big SMP

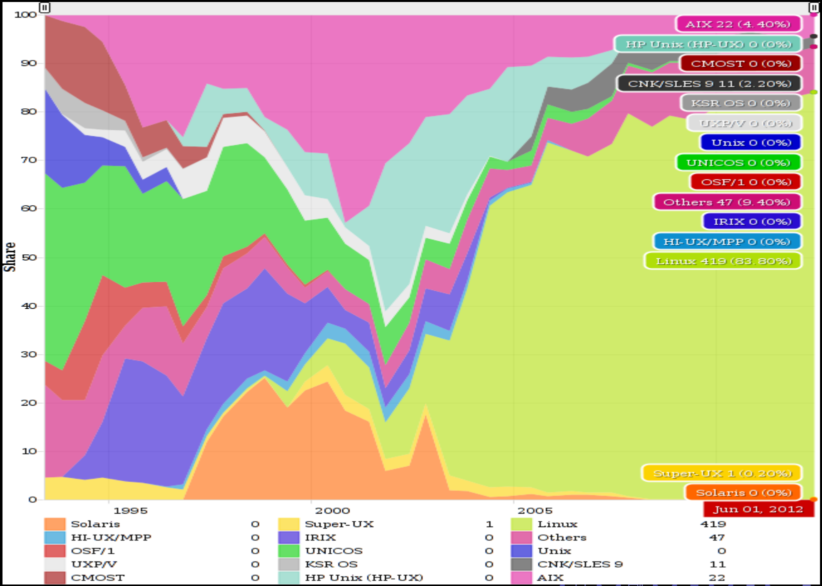
Types of parallel machine architectures (II)



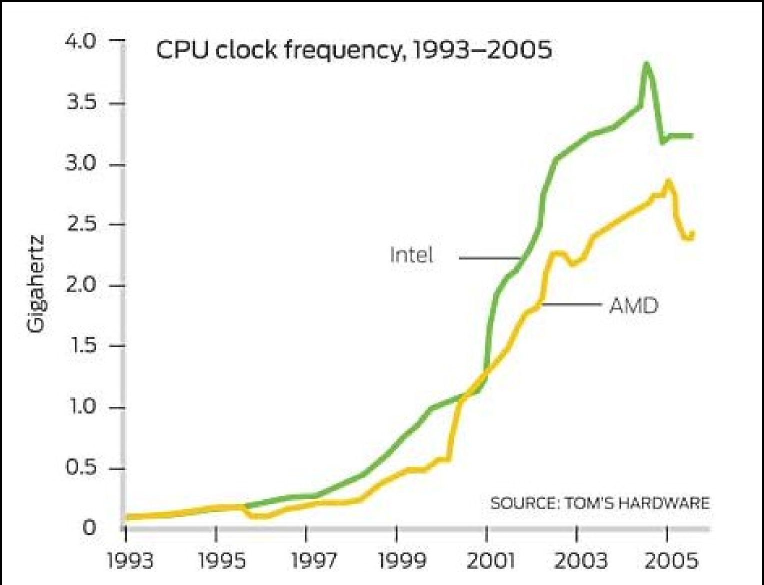
Processor families



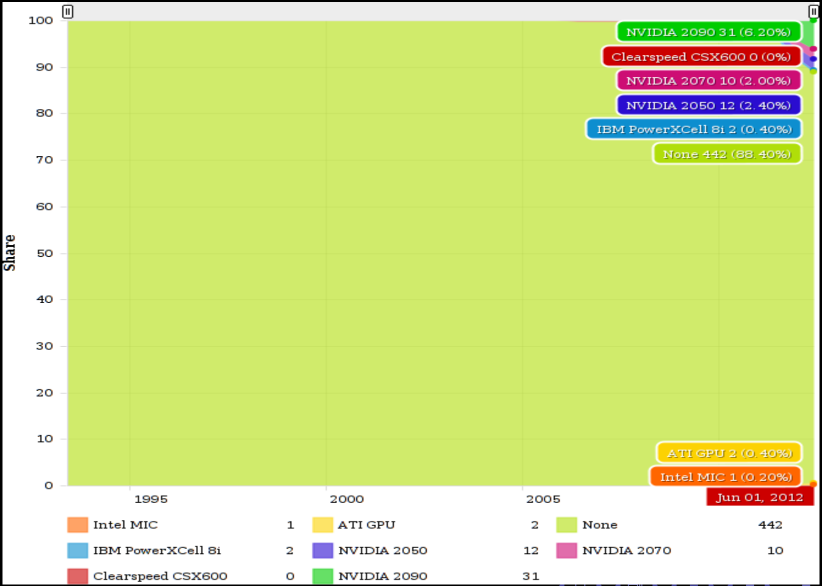
Operating system families



processor speed



Accelerator/co-processor evolution



Green 500

- Huge issues with electrical consumption and... heat in data-centers ☹
- 6 MW $\approx$ 600 €/h, $\approx$ 5 M€/year...
- Future with energy efficiency
- $\leadsto$ New ranking list
  - TOP Green500: most powerful supercomputers running the Linpack benchmark ranked by energy efficiency
  - Little Green500: most energy-efficient supercomputers achieving at least 9 tflops on the linpack benchmark
  - Open Green500: exploratory list for energy-efficient supercomputers achieving more than 9 tflops on linpack however they wish
  - HPCC Green500: exploratory lists for most energy-efficient supercomputers running the HPCC benchmark

`http://www.green500.org`

Tendency

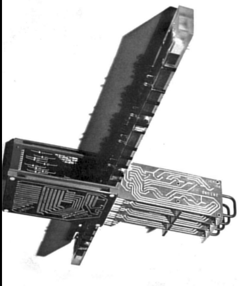
- Processor clock: no longer improvement
- Moore's law gives more transistors yet
  - Even more parallelism: *many*cores
  - Heterogeneous architectures with specialized power-efficient accelerators : GPGPU, Cell, vectoriel/SIMD, FPGA
- Compilers always behind... ☹

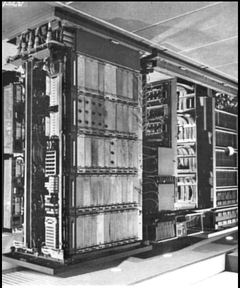
Outline

Multicores strike back... (I)

Gamma 60 from Compagnie des Machines Bull

- Increase the performance
- 100 kHz memory clock
- *Heterogeneous* multicore because the memory is too... fast! ☺
- 24-bit words, 96 KiB of core memory
- Punch-cards with ECC, magnetic tapes, magnetic drums
- Highly integrated logic in 1-mm germanium bipolar lithography ☺





Multicores strike back... (II)

- Gamma 60 multithread programming with `SIMU` ($\approx$ fork) & `CUT` (launch a program on a functional unit) instructions
- Synchronization barrier by concurrent branching on a same target
- Scheduling of threads based on a queue per functional unit stored just... inside the code after each `CUT` instruction!
- Optional hardware critical section on subprograms (cf. `synchronized` of Java)

Multicores strike back... (III)

- Installation around 1959
- Already hard to program since the concepts was not here, at most the (grand-)parents of anyone who were to know them...

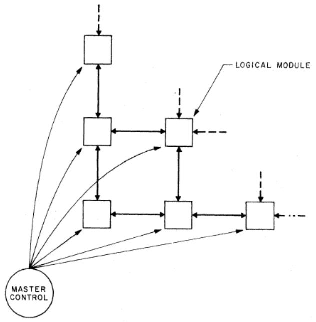
http://www.feb-patrimoine.com/projet/gamma60/gamma_60.htm

GPGPUs: just more integrated... (I)

- The “Distributed Computer”
  - Toward computing on spatial data : pattern recognition, mathematical morphology...
  - Massive parallelism to reduce the cost
  - SIMD

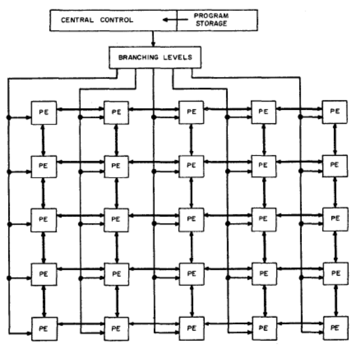
S. H. Unger. Â« A Computer Oriented Toward Spatial Problems. Â» *Proceedings of the IRE*. p. 1744–1750.

oct. 1958



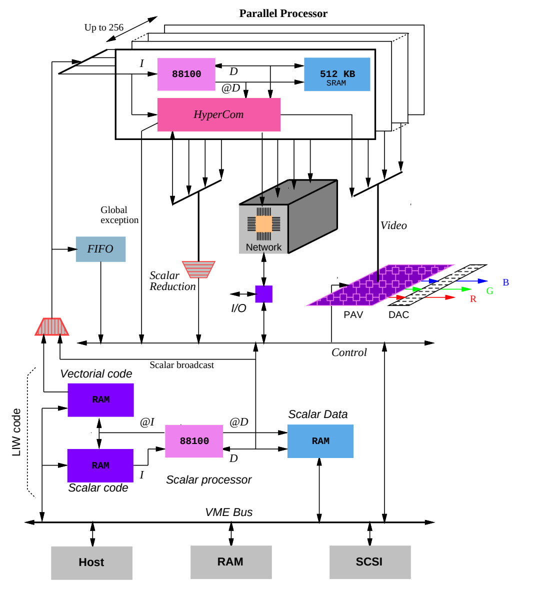
GPGPUs: just more integrated... (II)

- SOLOMON
  - Target application: “data reduction, communication, character recognition, optimization, guidance and control, orbit calculations, hydrodynamics, heat flow, diffusion, radar data processing, and numerical weather forecasting”
  - Diode + transistor logic in 10-pin TO5 package



Daniel L. Slotnick. Â« The SOLOMON computer. Â» *Proceedings of the December 4-6, 1962, fall joint computer conference*. p. 97–107. 1962

POMP & PompC @ LI/ENS 1987–1992



HyperParallel Technologies (1992–1998)

- Parallel computer
- DEC Alpha 21064
- FPGA-based 3D-torus network
- HyperC (follow-up of PompC @ LI/ENS Ulm)
  - PGAS (Partitioned Global Address Space) language
  - An ancestor of UPC...

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹

- Killed by niche market + side effect of French bank scandal... ☹
- American subsidiary with dataparallel analytics application acquired by Yahoo! in 1998, $8M
- Closed technology $\rightsquigarrow$ lost for customers and... founders ☹
- But some concepts still in use in modern GPU: Ronan KERYELL & Nicolas PARIS. « Activity Counter : New Optimization for the Dynamic Scheduling of SIMD Control Flow. » ICPP 1993

HyperParallel Technologies (1992–1998)

- Parallel computer
- DEC Alpha 21064
- FPGA-based 3D-torus network
- HyperC (follow-up of PompC @ LI/ENS Ulm)
  - PGAS (Partitioned Global Address Space) language
  - An ancestor of UPC...

Quite simple business model

- Customers need just to rewrite all their code in HyperC ☺
- Difficult entry cost... ☹

- Killed by niche market + side effect of French bank scandal... ☹
- American subsidiary with dataparallel analytics application acquired by Yahoo! in 1998, \$8M
- Closed technology ⇝ lost for customers and... founders ☹
- But some concepts still in use in modern GPU: Ronan KERYELL & Nicolas PARIS. « Activity Counter : New Optimization for the Dynamic Scheduling of SIMD Control Flow. » ICPP 1993

Parallelism is the only way to go

Just wait for $\mathcal{O}(1)$ years...

Outline

Software evolution

Up to now...

- Assembly languages
- High-level language for machines *à la* von NEUMANN (Fortran, C...)
- Object-oriented programming for comparability, malleability and maintainability of huge software projects
- Libraries with components, tools, design-patterns
- Specifications, modeling, methodologies, tests...

High performance?

... Trust MOORE's law ☺

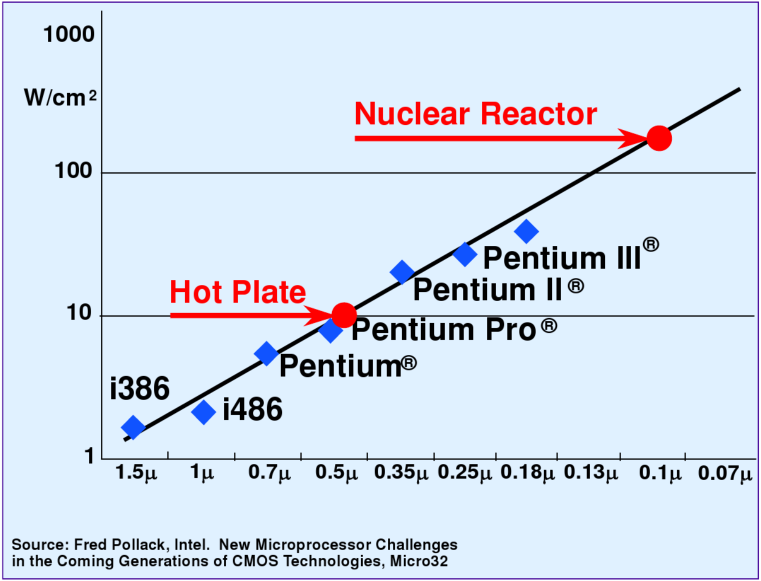
Processor-oblivious programmers...

- High-level languages far from hardware (even worse with JVM, CLI...)
- Abstraction that gave liberty and creativity ☺
- Future to Web 3.0! ☺
- Speed? Keep that for workers in the shade (MOORE's law...): an antique program run quite quicker on any modern processor! ☺

⇝ A programmer can ignore anything about processors

$\exists^?$ serious computer architecture in French graduate school? ☹

Power density



The end of sequential performance increasing...

- From $\times 2/1.5$ year down to $\times 2/\approx 5$ years... ☹
- But still needs for performance
  - μηδὲν ἄγαν
  - Bigger data
  - More functionalities/€
  - More functionalities/W

Only solution: parallelism...

...and keep a good temper: « comparability, malleability and maintainability, portability... »

How to face this?

Present motivations: reinterpreting Moore's law (I)

The good news ☺

- Number of transistors still increasing
- Memory storage increasing (DRAM, FLASH...)
- Hard disk storage increasing
- Processors (with captors) everywhere
- Network is increasing

- The bad news ☹
  - Transistors are so small they leak... Static consumption
  - Superscalar and cache are less efficient compared to transistor budget
  - Storing and moving information is expensive, computing is cheap: change in algorithms...
  - Light's speed has not improved for a while... Hard to reduce latency
    - Chips are too big to be globally synchronous at multi GHz ☹

SILKAN

Present motivations: reinterpreting Moore's law (II)

- pJ and physics become very fashionable
- Power efficiency in $\mathcal{O}(\frac{1}{f})$
  - Transistors cannot be used at full speed without melting ☹
  - Heat
- I/O and pin counts
  - Huge time and energy cost to move information outside the chip ☹

- Rotating hard disk with 1D density d increase
  - Storage in $\mathcal{O}(d^2)$
  - But track speed only $\mathcal{O}(d)$
  - $\rightsquigarrow$ Reading all the disk in $\mathcal{O}(\frac{1}{d})$ ☹

Heterogeneous parallelism

Parallelism is the only way to go...

- Research is just crossing reality!
- Scaring... ☹
- Exciting! ☺

No one size fit all...

Future will be heterogeneous

The “Software Crisis”

Edsger DIJKSTRA, 1972 Turing Award Lecture, « The Humble Programmer »

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”

`http://en.wikipedia.org/wiki/Software_crisis`

But... it was before parallelism democratization! ☹

Outline

HPC Project emergence

- $\approx$ 2006 : power consumption
  - $\rightsquigarrow$ Multicores and programmable GPU almost mainstream!

Time to be back in parallelism!

Yet another start-up... ☺

- Friends that met $\approx$ 1990 at the French Parallel Computing military lab SEH/ETCA
- Later became researchers in Computer Science, CINES director and ex-CEA/DAM ($\approx$ ASCI in US), venture capital and more: ex-CEO of Thales Computer, HP marketing...
- HPC Project launched in December 2007
- Now $\approx$ 50+ colleagues in France (Montpellier, Meudon), Canada (Montréal with Parallel Geometry) & USA (Wild Systems in Santa Clara, CA)
- Just closing our 3rd fund raising
- Fusion with ARION for expertise in critical embedded systems $\rightsquigarrow$ SILKAN

SILKAN business

Wide expertise in parallel computing & high-end simulation

- Engineering services (application development for parallel & embedded systems)
- Parallel simulator architectures (military training...)
- Libraries for dense & sparse linear algebra on multiple heterogeneous accelerators: Wild Runner
- Parallelizing tools for Scilab, C & Fortran to OpenMP, CUDA & OpenCL: Wild Cruncher & Par4All
- Professional training (parallel programming, OpenMP, MPI, TBB, CUDA, OpenCL...)

Outline

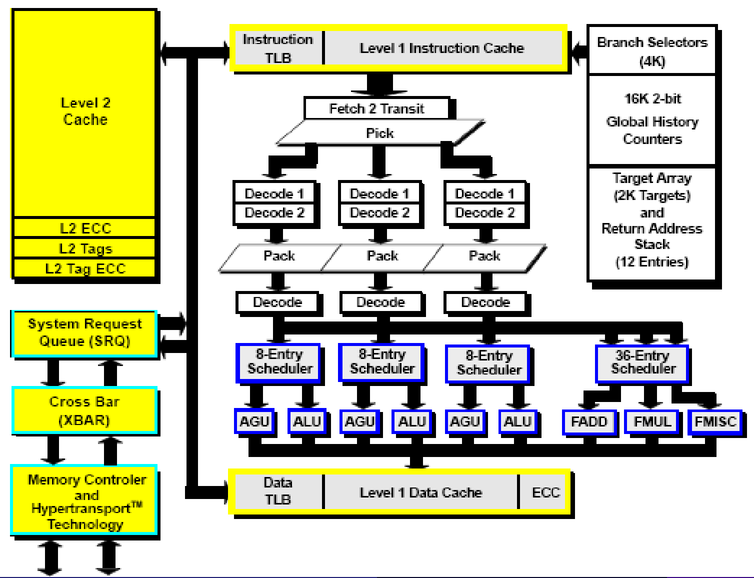
Outline

AMD Opteron 6180 (2011) (I)

- *x*86-compatible instruction set
- 64-bit mode that doubles also register numbers
- Out-of-order superscalar execution with 9 instructions/cycle
  - 3 integer instructions
  - 3 address generators
  - 3 floating computation operators (+, *, memory)
- Opteron 6180 SE, 02/2011, $1514
  - 12 cores @ 2.5 GHz, 512 KB/core L2 cache
  - 2 $\times$ 6 MB L3 cache
  - 21 GB/s DDR3 memory
  - 4 x 6.4 GT/s HyperTransport
  - 1.8 Gtr 45 nm 692 mm^2
  - 140 W

AMD Opteron 6180 (2011) *(II)*

http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs



IBM Power 7 (2010)

- 4 chips per quad-chip module
- 8 cores per chip @ 4.0 GHz (4.25 GHz in TurboCore mode with 4 cores)
- 1.2 Gtr 45 nm SOI process, 567 mm^2
- 4 SMT threads per core
- 32I+32D kB L1 cache/core
- 256 kB L2 cache/core
- 32 MB L3 cache in eDRAM
- 100 GB/s DDR3 memory
- 12 execution units per core:
  - 2 fixed-point units
  - 2 load/store units
  - 4 double-precision floating-point units
  - 1 vector unit supporting VSX
  - 1 decimal floating-point unit
  - 1 branch unit
  - 1 condition register unit

SILKAN

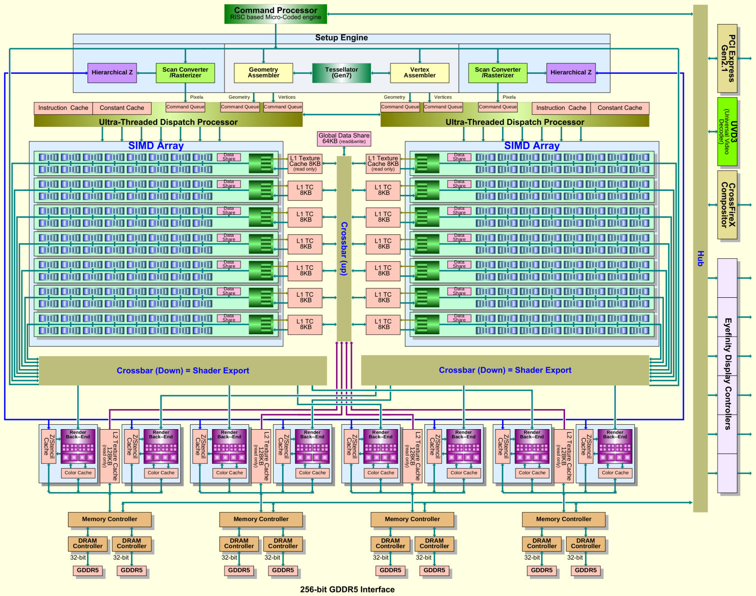
Off-the-shelf AMD/ATI Radeon HD 6970 GPU

- 2.64 billion 40nm transistors
- 1536 stream processors @ 880 MHz, 2.7 TFLOPS SP, 675 GFLOPS DP
- + External 1 GB GDDR5 memory 5.5 Gt/s, 176 GB/s, 384b GDDR5
- 250 W on board (20 idle), PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL
- ∃ Radeon HD 6990 double chip card

More integration:

- Llano APU (FUSION Accelerated Processing Unit) : x86 multicore + GPU 32nm, OpenCL

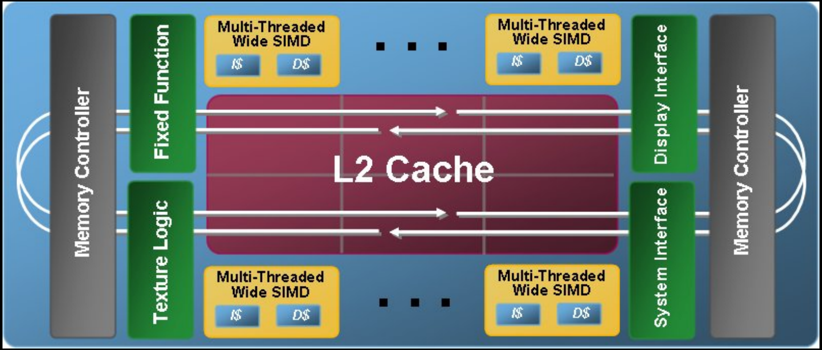
Radeon HD 6870 — big picture



HPC Overview

Intel Xeon Phi (2012)

- More simpler processors
- Start with old 0,8 μm Pentium
- Pack 54 processors with 22nm technology
- Superscalar in order
- 8 GB RAM
- SIMD/vector instructions on 512 bit operands



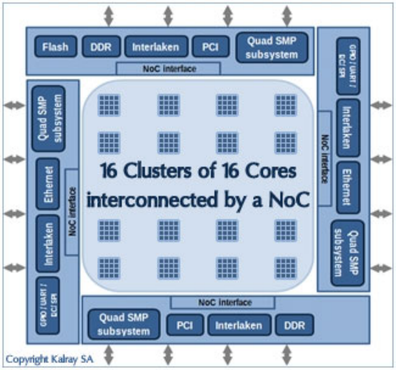
ARM yourself (I)

- Do some computations where the captors are...
- Smartphone and other sensor networks
- Trade-off between communication energy and inside/remote computations
- Texas Instrument OMAP4470 announced on 2011/06/02
  - 2 ARM Cortex-A9 MPCores @ 1.8GHz with Neon vector instructions
  - 2 ARM Cortex-M3 cores (low-power and real-time responsiveness, multimedia, avoiding to wake up the Cortex-A9...)
  - **SGX544 graphics core with OpenCL 1.1 support**, with 4 USSE2 core @ 384 MHz producing each 4 FMAD/cycle: 12.3 GFLOPS
  - 2D graphics accelerator
  - 3 HD displays and up to QXGA (2048x1536) resolution + stereoscopic 3D
  - Dual-channel, 466 MHz LPDDR2 memory

ARM yourself (II)

- $\rightsquigarrow$ Current course to have non-x86 servers based on ARM...
- $\exists$ Experiments on low power clusters
- CARMA (CUDA Development Kit for ARM) from nVidia for Tegra-3 `http://www.nvidia.com/object/carma-devkit.html`
- Think to evaluate power consumption on your application

MPPA by Kalray (the French touch!) (I)



- 256 VLIW processors per chip 16 clusters of 16 processors)
- Shared memory and NoC with DMA
- 28 nm CMOS technology, $\approx$ 5 W @ 400 MHz
- FPU 32/64 bits IEEE 754: 205 GFLOPS SP, 1 TOPS 16 bits
- 2$\times$ 64-bit DDR3 memory controllers for high bandwidth main memory transfers
- 2$\times$ 40 Gb/s or 8$\times$ 10 Gb/s) Ethernet controller

MPPA by Kalray (the French touch!) (II)

- $2\times$ 8-lane PCI Express Gen 3
- $4\times$ 4–8-lane Interlaken interfaces for multi-MPPA chip system integration (8 MPPA/PCIe board) or connection to external FPGAs, I/O...
- Linux or bare metal with AccessCore library
- Multi-core compiler (gcc 4.5), simulator, debugger (gdb), profiler
- Eclipse IDE
- Programming from high-level C-based language
- AccessCore library

FPGA (I)

- Field-programmable gate array with bitstream configuration in memory
- Xilinx Virtex 7
  - 2M logic cells (6-LUT), 28 nm
  - 85Mb block RAM
  - 5280 DSP slices (6.7 TFMA/s)
  - 96 transceiver @ 28Gb/s: 2.8 Tb/s
  - PCIe gen3 $\times 8$
  - 1200 pins
- Radar, communications, HPC, datamining, bioinformatics...

FPGA (II)

- VHDL-to-bitstream compiler... but **hard work**
- Dynamic partial reconfiguration
- ∃ C-to-VHDL compilers
  - Riverside Optimizing Compiler for Configurable Computing (ROCCC): open source
  - Impulse C
  - Catapult C
  - Cynthesizer
  - ...

Convey HC-1^{ex}

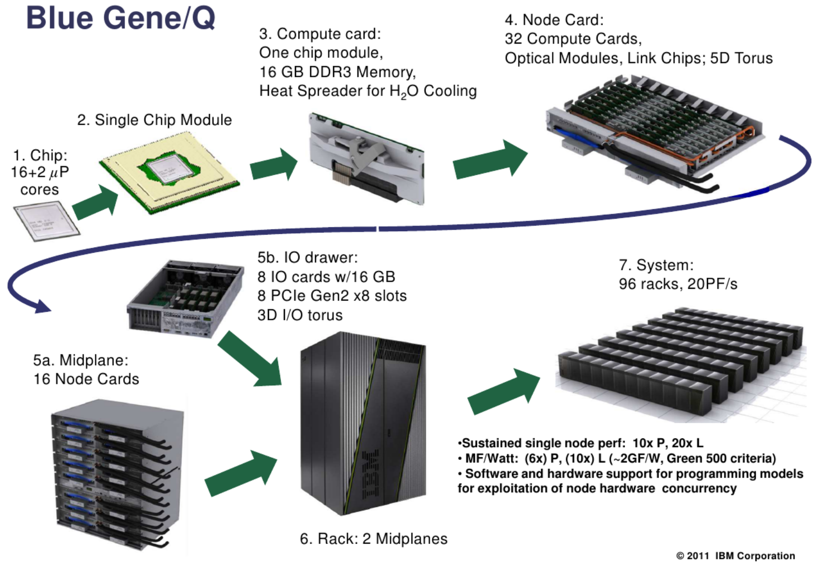
- Intel Xeon quad-core @2.13 GHz with 128 GB
- 4 Xilinx Virtex 6 LX760 FPGA with 128 GB DDR2
- 1520 W in 3U rack
- Linux
- Various instruction sets (“personality”)
- Fortran/C/C++ vectorizing compiler replacing complex instructions by FPGA vectorized implementation
- Possible to design its own instruction set (ROCCC...)
- 401$\times$ speed-up on Smith-Waterman algorithm compared to *x*86

Outline

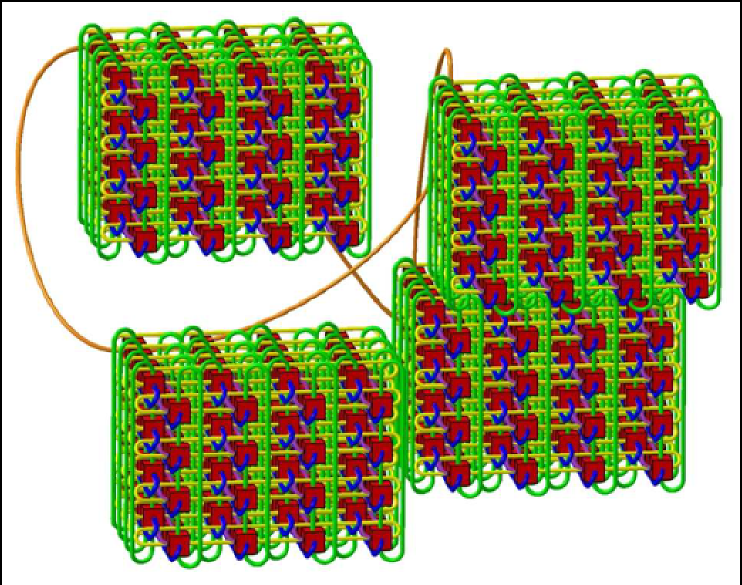
IBM BlueGene/Q *(I)*

- Top 1 in 2012 with 1 572 864 processors
- 16 PEs + 1 service + 1 spare with SIMD 4 double precision, 205 GFLOPS @ 55W
- 32 MB eDRAM cache
- 16 GB DDR3 RAM @ 42.6 GB/s
- 5D torus network
  - 2 GB/s/link in each direction
  - Diameter of 32 for a 20 PFLOPS machine
  - 80 ns/3μs latency
  - Hardware collective operations
  - RDMA
  - Transactional memory (multiple versions in cache)

IBM BlueGene/Q (II)



IBM BlueGene/Q 5D torus network



Outline

Expressing/finding parallelism?

Hypothesis

There is some parallelism exploitable in the application...

- Solution libraries
  - Need to fit your application
- New parallel languages
  - Rewrite your applications...
- Extend sequential language with #pragma
  - Nicer transition
  - Need sequential code expressing some parallelism
- Hide parallelism in object oriented classes
  - Restructure your applications...
- Use magical automatic parallelizer

Outline

OpenMP

- "Multithread for dummies" ☺
- Target SMP (shared memory processors)
- No need to use system threads, but for complex system programming
- Idea: powder a program with directives to help the compiler to parallelize
- Philosophy: `#pragma`tism with aestheticism
- If no directive, no parallelism (*a priori*)
- Directive ≡ sworn statement
- Supported languages: Fortran & C/C++

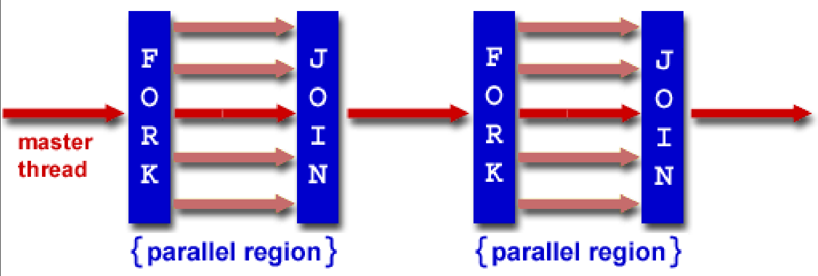
`http://openmp.org`

SILKAN

OpenMP execution model (I)

`http://openmp.org/wp/openmp-specifications`

- Parallel SPMD execution based on *fork/join*



- Implicit or explicit thread creation with *directives*

OpenMP execution model (II)

- Trick: an OpenMP program can be executed as
  - Sequential program
  - Parallel program

  $\leadsto \nearrow$ portability, zero exit cost! ☺
- Threads are created in `parallel` sections and stopped at the end with a *barrier*
- Synchronization constructions and with library functions
- Execution environment controlled by environment variables and library functions

Example

```
/* This is executed by several threads in // */
#pragma omp parallel for
  for (i=0; i<n; i++)
      // Iterations are distributed between the threads
      x[i] += y[i];
// Implicit synchronization here
```

Task with OpenMP 3.0

- Add concept of explicit tasks ≡ piece of program executed on a thread
- Task created with `task` construction (TBB, Cilk++...)

```c
#pragma omp parallel
{
   #pragma omp single private(p)
   {
    p = listhead;
    while (p) {
       #pragma omp task
       {
               process (p);
       }
       p=next(p);
     }
   }
}
```

- Extensions envisioned (`target`, OpenACC...)

SILKAN

Programmation CUDA (I)

- Data-parallel extension to a C++ subset
- Target nVidia GPU and *x*86 multicores
- 2-level parallelism: threads in blocks of threads + block-tiling
- In a block of threads : communication through shared memory and synchronization via `__syncthreads()`
- Complex heterogeneous memory layout (GPU...)

```
__global__ void
add_matrix_gpu(float *a, float *b, float *c, int N) {
  int i=blockIdx.x*blockDim.x+threadIdx.x;
  int j=blockIdx.y*blockDim.y+threadIdx.y;
  int index =i+j*N;

  if( i < N && j < N)
    c[index]=a[index]+b[index];
}

void main() {
  float ha[N][N], hb[N][N], hc[N][N];
```

Programmation CUDA (II)

```
    /* Allocate array on the GPU with cudaMalloc */
    float *a, *b, *c;
    cudaMalloc((void **) &a, sizeof(float)*N*N);
    cudaMalloc((void **) &b, sizeof(float)*N*N);
    cudaMalloc((void **) &c, sizeof(float)*N*N);

    cudaMemcpy(a, ha, sizeof(float)*N*N, cudaMemcpyHostToDevice);
    cudaMemcpy(b, hb, sizeof(float)*N*N, cudaMemcpyHostToDevice);

    // Décrit le pavage des itérations (strip-mining bidimensionnel)
    dim3 dimBlock (blocksize,blocksize);
    dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
    add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
    cudaMemcpy(c, hc, sizeof(float)*N*N, cudaMemcpyDeviceToHost);
}
```

- Need some heavy code restructuring

OpenCL (I)

- Language based on a C_{99} subset
- Started by Apple to *unify* parallel use (multicores, GPGPU...)
  ⇝ similar to OpenGL & OpenAL
- Followed by AMD/ATI and nVidia
- Data-parallelism and control-parallelism (1–3-dimensions) according to targets
- Kernel oriented computations on streams
- Complex split memory model (GPGPU...)
- New types (vectors, images...)

SILKAN

OpenCL (II)

```
/* This kernel computes FFT of length 1024.
   The 1024 length FFT is decomposed into calls to a radix 16
   function, another radix 16 function and then a radix 4 function */

__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;
  globalLoads(data, in, 64); // coalesced global reads
  fftRadix16Pass(data);      // in-place radix-16 pass
  twiddleFactorMul(data, tid, 1024, 0);
  // local shuffle using local memory
  localShuffle(data, sMemx, sMemy, tid,
               (((tid & 15) * 65) + (tid >> 4)));
  fftRadix16Pass(data);                // in-place radix-16 pass
  twiddleFactorMul(data, tid, 64, 4); // twiddle factor multiplication
  localShuffle(data, sMemx, sMemy, tid,
               (((tid >> 4) * 64) + (tid & 15)));
  // four radix-4 function calls
  fftRadix4Pass(data); fftRadix4Pass(data + 4);
  fftRadix4Pass(data + 8); fftRadix4Pass(data + 12);
```

OpenCL (III)

```
  // coalesced global writes
  globalStores(data, out, 64);
}
[...]
// create a compute context with GPU device
context = clCreateContextFromType(CL_DEVICE_TYPE_GPU);
// create a work-queue
queue = clCreateWorkQueue(context, NULL, NULL, 0);
// allocate the buffer memory objects
memobjs[0] = clCreateBuffer(context, CL_MEM_READ_ONLY |
CL_MEM_COPY_HOST_PTR,
                             sizeof(float)*2*num_entries, srcA);
memobjs[1] = clCreateBuffer(context, CL_MEM_READ_WRITE,
                             sizeof(float)*2*num_entries, NULL);
// create the compute program
program = clCreateProgramFromSource(context, 1,
                                    &fft1D_1024_kernel_src, NULL);
// build the compute program executable
clBuildProgramExecutable(program, false, NULL, NULL);
// create the compute kernel
kernel = clCreateKernel(program, "fft1D_1024");
// create N-D range object with work-item dimensions
global_work_size[0] = n;
local_work_size[0] = 64;
```

OpenCL (IV)

```
range = clCreateNDRangeContainer(context, 0, 1,
                                 global_work_size, local_work_size);
// set the args values
clSetKernelArg(kernel, 0, (void *)&memobjs[0], sizeof(cl_mem), NULL);
clSetKernelArg(kernel, 1, (void *)&memobjs[1], sizeof(cl_mem), NULL);
clSetKernelArg(kernel, 2, NULL,
               sizeof(float)*(local_work_size[0]+1)*16, NULL);
clSetKernelArg(kernel, 3, NULL,
               sizeof(float)*(local_work_size[0]+1)*16, NULL);
 // execute kernel
clExecuteKernel(queue, kernel, NULL, range, NULL, 0, NULL);
```

- Need a lot of code restructuring
- ∃ C++ wrappers to simplify house-keeping code

Outline

Message Passing Interface (MPI) *(I)*

- "Message Passing for dummies" ☺
- Communication library available for many operating systems and language
- $\rightsquigarrow$ Highly portable
- Lower compatibility level for parallel programming: possible to use MPI on SMP machines...
- Model: independent processes can exchange data through messages
- Point-to-point, broadcast and collective operations
- Single-sided communications (RDMA)

`http://www.mpi-forum.org`

Mathematical libraries (I)

- Libraries often optimized by hardware vendors
  - Intel Math Kernel Library (MKL)
  - AMD Performance Library ($\rightsquigarrow$ free Framewave)
- FFT : FFTW (*Fastest Fourier Transform in the West*, in C, generated from OCaml)...
- Many linear algebra libraries
  - BLAS (*Basic Linear Algebra Subprograms*) & PBLAS
  - LAPACK (*Linear Algebra PACKage*)
  - ScaLAPACK : SPMD version with MPI
  - SuperLU : direct solving of big sparse systems
  - PETSc (*Portable, Extensible Toolkit for Scientific Computation*): wide spectrum, above MPI

Outline

SILKAN

Classes

- If an object-oriented program uses parallelized classes $\rightsquigarrow$ parallel program!
- $\exists$ parallel Versions of STL, often Open Source (PaSTel @ INRIA...)
- Higher level libraries often already parallelized (databases, computer graphics...)

Thread Building Blocks (TBB) *(I)*

`http://en.wikipedia.org/wiki/TBB` from Intel

- Template library (*à la* STL)
- Open and commercial Versions
- Algorithms (`for`, `reduce`, `pipeline`, `scan`...), containers, memory allocators, mutual exclusion, atomic operations, schedulers, profiling...
- Work stealing between tasks
- Orthogonal to OpenMP & MPI

Thread Building Blocks (TBB) (II)

```
class ApplyFoo {
  float* const my_a;
public:
  ApplyFoo(float* a) : my_a(a) {};
  void operator() (const tbb::blocked_range<size_t>&r) const {
    for (size_t i=r.begin(); i != r.end(); ++i)
      Foo (my_a[i]);
  }
}

void ParallelApplyFoo(float a[], size_t n) {
  tbb::parallel_for (
    tbb::blocked_range<size_t>(0,n),
    ApplyFoo(a),
    tbb::auto_partitioner()
  );
}
```

- Need a deep code restructuring if an application is not in a STL spirit

ArBB (ex-Ct) : C/C++ for Throughput Computing (I)

- Array Building Blocks, fusion of Intel Ct language and technology from RapidMind
- C++ classes from Intel for its Terascale project
- Add « data collection » `TVEC` container
- Data-parallelism to deal easily with a lot of data
- Data-parallelism operations as in Nesl, PompC, HyperC... languages
  - Vector 1-to-1 functions
  - Collective communications: reductions, scan...
  - Permutations: scatter/gather, (un)packing, shift, partitions, butterfly...

ArBB (ex-Ct) : C/C++ for Throughput Computing (II)

```
TVEC<F32> colorConvert(TVEC<F32> rchannel, TVEC<F32> gchannel,
                       TVEC<F32> bchannel, TVEC<F32> achannel,
                       F32 a0, F32 a1, F32 a2, F32 a3) {
  return (rchannel*a0 + gchannel*a1 + bchannel*a2 + achannel*a3);
}

TVEC<F32> Convolve2D3x3(TVEC<F32> pixels, TVEC<F32> kernel) {
  TVEC<F32> respixels;
  // directions[m][n] is a constant TVEC of size
  // with values {m-1, n-1}
  respixels += shiftPermute(pixels, directions[0][0])*kernel[0][0];
  respixels += shiftPermute(pixels, directions[0][1])*kernel[0][1];
  respixels += shiftPermute(pixels, directions[0][2])*kernel[0][2];
  respixels += shiftPermute(pixels, directions[1][0])*kernel[1][0];
  respixels += pixels*kernel[1][1];
  respixels += shiftPermute(pixels, directions[1][2])*kernel[1][2];
  respixels += shiftPermute(pixels, directions[2][0])*kernel[2][0];
  respixels += shiftPermute(pixels, directions[2][1])*kernel[2][1];
  respixels += shiftPermute(pixels, directions[2][2])*kernel[2][2];
  return respixels
}
```

ArBB (ex-Ct) : C/C++ for Throughput Computing (III)

- Task graph to execute different data-parallel computations with some control parallelism
- Need some code restructuring to fit the data-parallel model

C++0x... 2011 (I)

- Parallelism and synchronization are mainstream ⤳ inclusion in last C++ version
- `#include <atomic>`
  - Atomic load, atomic store with different possible memory consistency models
  - Atomic side effect operators: =, ++, -=...
  - Compare-exchange
  - Fetch-and-op
  - Some lock-free implementations
  - Memory fence
  - ...
- `#include <thread>`
  - Describe threads to be mapped one-to-one with OS threads
  - Creation
  - Join (wait for another thread to complete)
  - Get hardware concurrency hint

C++0x... 2011 (II)

- Sleep
- Yield

- **#include** `<mutex>`
  - Lock, recursive or not
  - Unlock
  - `try_lock()`, potentially with timeout
- **#include** `<condition_variable>`
  - Wait for a condition given by another thread
  - `wait()`
  - `notify_one()`, `notify_all()`
- **#include** `<future>`
  - Can get results from function execution (typically in another task)

Boost C++ (I)

- **#include** `<boost/thread.hpp>`
  - Pre-C++0x implementation
  - Contains in the same package also mutex, condition variables and future

  `http://www.quantnet.com/cplusplus-multithreading-boost`
- **#include** `<boost/mpi.hpp>`
  - C++ wrapping of Message Passing Interface
  - Better integration with STL spirit than native MPI C++ support
- BLAS wrapping in `ublas`

OpenACC (I)

- Extend the concept of OpenMP `#pragma`
- Target SMP, vector extensions (AVX, NEON...), GPU
- Standard developed by Cray, CAPS, Nvidia and PGI

OpenACC (II)

```fortran
!$acc data copyin(A), create(Anew)
iter = 0
do while ( err .gt. tol .and. iter .gt. iter_max )
  iter = iter + 1
  err = 0.0
!$acc kernels loop reduction( max:err ), gang(32), worker(8)
  do j=1,m
    do i=1,n
      Anew(i,j) = 0.25 * ( A(i+1,j ) + A(i-1,j ) &
                         A(i, j-1) + A(i, j+1) )
      err = max( err, abs(Anew(i,j)-A(i,j)) )
    end do
  end do
!$acc end kernels loop
  if( mod(iter,100).eq.0 ) print*, iter, err
!$acc parallel
  A = Anew
!$acc end parallel
end do
!$acc end data
```

SILKAN

Outline

Automatic parallelization (I)

- Major research failure from the past...
- ... We used to work on automatic parallelization & HPF Fortran compilation
- Untractable in the general case ☹
- But automatic parallelization technology widely used locally in main compilers
- Bad sequential programs? GIGO: Garbage In-Garbage Out...
- To use #pragma, // languages or classes: cleaner sequential program or algorithm first...

Automatic parallelization (II)

Agent SMITH:

Never send a human to do a machine's job.

In *Matrix* (Andy & Larry WACHOWSKI, 1999)

Automatic parallelization (III)

- ... and then automatic parallelization can often work ☺
- $\rightsquigarrow$ Par4All = automatic parallelization + coding rules
- Often less optimal performance but better time-to-market

Basic Par4All coding rules for good parallelization (I)

- Same constraints as `for`-loop accepted in OpenMP standard
- `for ([int]` *init-expr*; *var relational-op b*; *incr-expr*)
  `statement`
- Increment and bounds: integer expressions, loop-invariant
- *relational-op* only <, <=, >=, >
- Do not modify loop index inside loop body
- Do not use `assert()` or compile with `-DNDEBUG` inside a loop. Assert has potential exit effect
- No `goto` outside the loop, `break`, `continue`
- No `exit()`, `longjump()`, `setcontext()`...
- Data structures
  - Pointers
    - Do not use pointer arithmetics

Basic Par4All coding rules for good parallelization (II)

- Arrays
  - PIPS uses integer polyhedron lattice in analysis
  - Do not use linearized arrays
  - Use affine reference in parallelizable code

```
// Good:
a[2*i-3+m][3*i-j+6*n]
// Bad (polynomial):
a[2*i*j][m*n-i+j]
```

- Do not use recursion
- Prototype of coding rules report on-line on `par4all.org`
- ... Useful not only for Par4All!

Outline

Stars-PM

- *Particle-Mesh* N-body cosmological simulation
- C code from Observatoire Astronomique de Strasbourg
- Use FFT 3D
- Example given in `par4all.org` distribution

Stars-PM time step

```
void iteration(coord pos[NP][NP][NP],
               coord vel[NP][NP][NP],
               float dens[NP][NP][NP],
               int data[NP][NP][NP],
               int histo[NP][NP][NP]) {
    /* Split space into regular 3D grid: */
  discretisation(pos, data);
    /* Compute density on the grid: */
  histogram(data, histo);
    /* Compute attraction potential
       in Fourier's space: */
  potential(histo, dens);
    /* Compute in each dimension the resulting forces and
       integrate the acceleration to update the speeds: */
  forcex(dens, force);
  updatevel(vel, force, data, 0, dt);
  forcey(dens, force);
  updatevel(vel, force, data, 1, dt);
  forcez(dens, force);
  updatevel(vel, force, data, 2, dt);
    /* Move the particles: */
  updatepos(pos, vel);
}
```

SILKAN

Stars-PM & Jacobi results

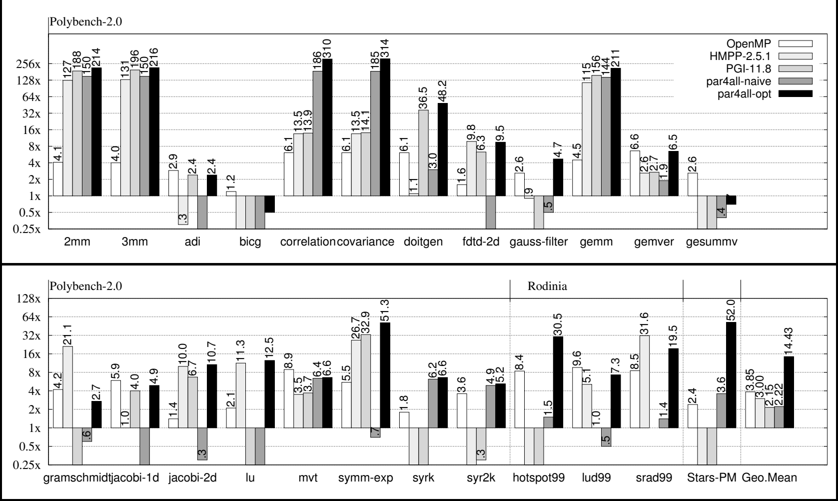
- 2 Xeon Nehalem X5670 (12 cores @ 2,93 GHz)
- 1 GPU nVidia Tesla C2050
- Automatic call to CuFFT instead of FFTW
- Time and speed-up for 150 iterations of Stars-PM

Speed-up	p4a	Simulation Cosmo. $128 \times 128 \times 128$	Jacobi
Sequential (time in s)	(gcc -O3)	98.4 s	24.5 s
OpenMP *6 threads*	--openmp	5.9	1.78
CUDA base	--cuda	3.13	0.36
Optim. comm.	+ --com-optim	11	3.8
Reduction Optim.	+ --atomic	46.9	6.4
Manual optim.	(gcc -O3)	54.7	

SILKAN

Benchmark results

With Par4All 1.2, CUDA 4.0, WildNode 2 Xeon Nehalem X5670 (12 cores @ 2.93 GHz) with nVidia C2050. [LCPC'2011]



From `par4all.org` distribution, in `examples/Benchmarks`

Outline

Scilab language

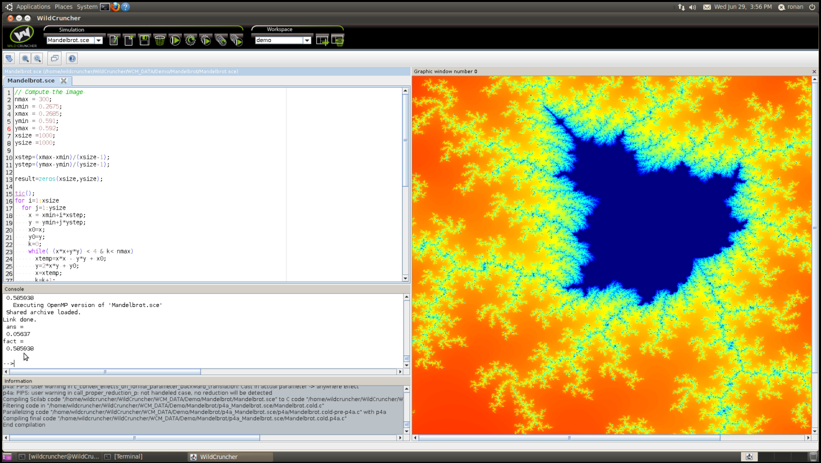
- Interpreted scientific language widely used like Matlab
- Free software
- Roots in free version of Matlab from the 80's
- Dynamic typing (scalars, vectors, (hyper)matrices, strings...)
- Many scientific functions, graphics...
- Double precision everywhere, even for loop indices (now)
- Slow because everything decided at runtime, garbage collecting
  - Implicit loops around each vector expression
    - Huge memory bandwidth used
    - Cache thrashing
    - Redundant control flow
- Strong commitment to develop Scilab through Scilab Enterprise, backed by a big user community, INRIA...
- HPC Project WildNode appliance with Scilab parallelization
- Reuse Par4All infrastructure to parallelize the code

Scilab & Matlab

- Scilab/Matlab input : *sequential* or array syntax
- Compilation to C code with our own compiler
- Parallelization of the generated C code
- Type inference to guess (crazy ☹) semantics
  - Heuristic: first encountered type is forever
- Speedup $>$ 1000 in some cases ☺
- Wild Cruncher: *x*86+GPU appliance with nice interface
  - Scilab — mathematical model & simulation
  - Par4All — automatic parallelization
  - //Geometry — polynomial-based 3D rendering & modelling
- Versions to compile to other platforms (fixed-point DSP...)

SILKAN

Wild Cruncher — Scilab parallelization IDE



SILKAN

Outline

Conclusion (I)

- No tool or language will solve all the issues...
- Often hierarchical solution for heterogeneous hardware
- Domain is maturing: many languages, libraries, applications, tools... Just choose the good one ☺
- Real codes are often not well written to be parallelized... even by human being ☹
- At least writing clean C99/Fortran/Scilab... code should be a prerequisite
- Take a positive attitude... Parallelization is a good opportunity for deep cleaning (refactoring, modernization...) ⤳ improve also the original code
- Open standards to avoid sticking to some architectures
- Need software tools and environments that will last through business plans or companies

Conclusion (II)

- Open implementations are a warranty for long time support for a technology (cf. current tendency in military and national security projects)
- Entry cost
- Exit cost! ☹
  - Do not loose control on *your* code and *your* data!
- We are hiring massively: developers and research engineers (parallelism, scientific computing, computer graphics, compilation, operating systems, simulation, QA...), PhD candidate (CIFRE), interns... on the various sites

SILKAN

SILKAN

SILKAN