Theme: Towards Reconfigurable High-Performance Computing
Lecture **10**

# Hybrid Platforms, Hybrid Programming?

**Iris Christadler, Leibniz Supercomputing Centre**
**Manfred Mücke, University of Vienna**
**Andrzej Nowak, CERN openlab**

Inverted CERN School of Computing, 3-5 March 2008





"Why is this happening? In the never-ending quest for more computational power, many in the industry already see the end in site for conventional multi-processors, multi-core architectures. After a while, just adding more processors to a system will have no effect. If a system has more cores than you have application threads, all the extra CPUs just become **Lilliputian space heaters**.

The heterogeneous approach offers greater efficiency by using specialized processing engines that can be matched more closely with different types of application code. A specialized chip, such as a GPU, an FPGA or a vector processor, can replace 100 conventional processors for certain types of codes. So the upside potential is enormous."

[http://www.hpcwire.com/hpc/897414.html]





# MULTICORES REVISITED



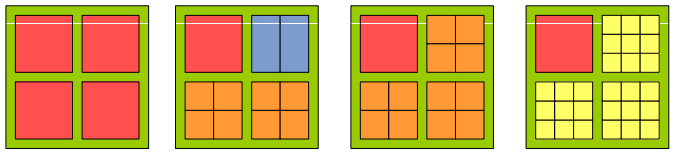


## Multicore Prospects revisited

- **Multicore designs will continue to dominate the computing landscape for at least several years**

**The multicore tradeoff**

**Large amounts of computing power will be available at your disposal, but an effort will be needed in order to put them to use**



## Supercomputing on Commodity Hardware – Outlook

- **X86 hardware is getting very powerful**
  - "Too powerful" some might say
  - Extremely versatile, very affordable
  - Adequate performance per Watt
- **Back to the mainframe days?**
- **What comes next after multicore?**
- **What about virtualization?**
  - Will we need to partition our resources?
  - Minor advantages in high efficiency scenarios – will this change?
- **GRID computing or tera/peta-scale homogenous computers?**





## Moving On

**Multicores will trigger some changes:**

- To use more CPUs efficiently, we need a distributed programming model (at least implicit data parallelism)
- To exploit hierarchical communication networks (on-chip, off-chip) we need some abstraction layer above MPI
- To be robust, we will need some run-time system (watch the MTBF for 1000+ Cores)

⇨ Every chip-manufacturer will invest in software (watch Intel)

⇨ This might be an enabling development for:
HPC on GRID
HPC on GPUs/FPGAs





## Mainstream Accelerators – GPUs

- **GPGPU is still a niche**
  - Partly because of the lack of proper tools
  - Partly because other custom solutions with a large overhead offer better FLOP/Watt performance
- **Graphics processing hardware will continue to evolve at a rapid pace**
  - New designs
  - Cores might get "heavier"
- **Graphics processing chip makers are listening to the scientific community**
- **Following developments in this area should be worthwhile**





## Multicores & FPGA/GPU

- **Multicores will (sooner or later) impose some (hopefully implicit) data-parallel programming model.**
  ⇨ Technology will migrate into standard compilers.
  ⇨ Efficient (application-wide?) datapath analysis.

⇨ **Once compilers are ready to assess distribution of computations over several cores, accelerators can be considered easily.**

⇨ **More unified tool-chain**

⇨ **FPGAs and GPUs could become the upgrade-path for tomorrow's PCs.**

## Multicores & GRID

**Currently, most GRID-jobs are single-CPU jobs!**
**This is only helpful for embarrassing parallel problems**

**⇨ How to execute MPI jobs efficiently on the Grid?**

- **Every MPI application implicitly assumes some (constant and uniform) computation/communication ratio.**
- **The bigger the differences within a system and between systems, the less efficient your application!**

Chip manufacturers
Grid Users

- **There is little difference between the challenges of distributing an application over 80 (on-chip) cores and 128 interconnected servers.**



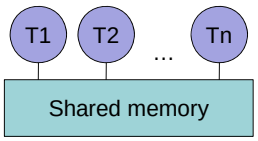


## NEW PROGRAMMING LANGUAGES

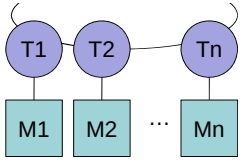




Shared Memory Model

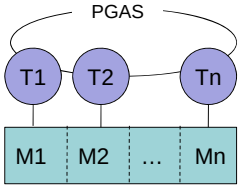



Distributed Memory Model




**Partitioned Global Address Space Languages**

**PGAS**








## PGAS Languages

- **They come in a triple**
  - Co-Array Fortran (CAF)
  - Unified Parallel C (UPC)
  - Titanium (Java)
- **They simplify parallel programming,** but are mainly “syntactic sugar for MPI”
- **They are most probably the best guess for new languages**
  - Compilers exist
  - CAF will be included in the next Fortran standard
  - CAF and UPC will be available e.g. on Cray XT5h
- **Their advantage:**
  - The compiler decides about parallelization

## Co-Array Fortran

- **"Fortran is not Fortran" ☺**
  - First CAF language definition is from 1998
  - Will be part of the next Fortran standard
  - Hopefully part of widely used HPC compilers around 2010
- **How does it work**
  - Each program consists of several *images* of your data
  - An image has its own set of data objects
  - An image runs on one core
  - The programmer can easily access data from other images
  - ➢ "Message passing" by hand is no longer necessary
- **Further information**
  [www.co-array.org]





## Unified Parallel C

- **Extension to the C standard**
  - many different compilers available
- **How does it work**
  - Very similar to CAF
  - One common global partitioned address space
  - But variables are physically assigned to one single processor
  - Programmer controls performance
    Critical decisions: Data layout and communication
- **Further information**
  [http://upc.gwu.edu/] >> Wiki
- **Titanium for Java programmers**





**High Productivity Computer Systems-Initiative**

## HPCS-LANGUAGES





## HPCS-Languages

- **High Productivity Computer Systems (HPCS)**
- **They come in a triple, too**
  - Fortress (SUN)
  - Chapel (Cray)
  - X10 (IBM)
- **They are very different**
  … but competition furthers the field
- **General remarks about the languages**
  - No compilers yet that fully support the language
  - No high-performance yet

"**HPCwire: Can you help us understand the big picture of the DARPA HPCS program?**
D. Post: [..] The productivity team has been doing detailed case studies of representative scientific and engineering code projects to identify the characteristics of application codes, the workflows for code development and production, "bottlenecks" and obstacles for code development and production, and "lessons learned" so that decisions by the productivity team and the vendors are based on real data rather than anecdotal data. The potential vendors are developing new computer languages and tools that improve productivity by allowing programmers to express parallelism at higher levels of abstraction. The "catch 22" issue with new languages is that no one will use the new language until it is mature, and it will never become mature unless it is used. This has led an effort to consolidate the language efforts of the vendors to produce a single new language that the community can adopt."

[http://www.hpcwire.com/hpc/893353.html]





**The promising new ideas..**

## STREAM PROGRAMMING





## Stream Programming Languages

- **What is stream processing**
  - Use just vector data types
  - Define (only) vector operations on them
- **Advantages**
  - A "stream" algorithm is (inherently) extremely parallel and can therefore be mapped to highly parallel devices
  - The program could stay portable and the compiler optimizes for the particular hardware
  - Enables latency vs. area trade-off
- **Stream Programming Languages:** PeakStream (acquired by Google in June 2007), RapidMind, Brook, CUDA, Intel Ct





## NVIDIA CUDA

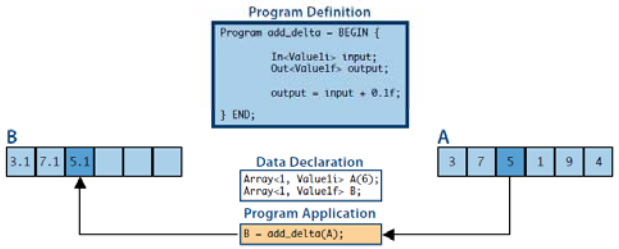
- **General purpose development kit for the G80 chip**
- **C supported, Open64 based compiler**
- **Includes BLAS and FFT libraries**
- **Deviations from the IEEE floating point standard**
- **Programming loadable kernels**
- **General example:**

```
int * dvalues;
CUDA_SAFE_CALL(cudaMalloc((void**)&dvalues, sizeof(int) * NUM));
CUDA_SAFE_CALL(cudaMemcpy(dvalues, values, sizeof(int) * NUM,
               cudaMemcpyHostToDevice));

bitonicSort<<<1, NUM, sizeof(int) * NUM>>>(dvalues);
```

## RapidMind

- Multicore and GPGPU development platform
- An API library for C++ with additions
- Numerous backends – x86, GPU, CELL




Graphics: Rapid Mind





## Intel Ct

- An experimental data parallel programming environment
- Designed to facilitate multicore programming and increase portability
- Best with vectors, sparse matrices, trees, linked lists
- Example:

```
CCtTVEC<double> sparseMatrixVectorProduct(
  CCtVEC<double> A, CCtVEC<int> rowindex,
  CCtVEC<int> cols, CCtVEC<double> v)
{
  CCtVEC expv = ctDistribute(v,cols);
  CCtVEC product = A*expv;
  return ctMultiReduceSum(product,rowindex);
}
```





## Intel STM

- A prototype version of the ICC C/C++ compiler
- Added transactional programming constructs
- Basic construct: __tm_atomic { *statements*; }
- Examples:

```
void foo(void){
   __tm_atomic {
   stmt1;
   stmt2;
   }
}
```

```
void func(void)
{
   try {
     __tm_atomic {
       stmt1;
       exception
       stmt2;
     } TM_HANDLER
   } catch/except {
       .........
   }
}
```





Hybrid Systems,

## HYBRID PROGRAMMING?

"And then there's the central problem of software. As difficult as it was (and is) to scale applications across more homogeneous processors, it will be significantly more complicated to slice up applications across a heterogeneous architecture. Heterogeneous-aware software (compilers, run-times, process/job schedulers, etc.) that intelligently maps the application code onto the available processor resources will be required for any sort of productive use of such systems."

[http://www.hpcwire.com/hpc/897414.html]





## Hybrid Programming

- **Hybrid programming =**
  1. Partition your application
  2. Choose a suitable language for each task/platform
  3. Link

  Famous example: Fortran (Maths on CPU) + MPI (Communication)

  Currently: CUDA for GPUs, VHDL for FPGAs
- **If you want to fully exploit your system: NO**
- **If you look back: YES**





## Pros & Cons

**Cons:**

- Manual partitioning
  Redo = rewrite application
- In distributed sytems, usually there is no single best partitioning strategy over a set of systems.
- How future-proof an application do you want ?
  Partitioning assumes fixed computation/communication ratio.
  Technology evolves fast (compare Ethernet vs. Infiniband)
  Do you know tomorrow's system?
- Target-specific languages prohibit automated evaluation and migration





## Pros & Cons

**Pros:**

- **Universal Compilers are too complex.**
- **Concentrate on what you can do best.**
- **Several small steps might be better than one disappointing big one.**
- **Small tools give room for understanding & research**

➢ **Only time will show**

...but maybe, requirements develop along similar lines

## HOW TO CONNECT THE WORLDS OF HPC AND FPGAS?





## Worlds Apart!?

- **HPC brings in very different requirements compared to embedded/DSP systems**
- **DSP: Speed is everything!**
  **Embedded: Balance price/power/size**
- **HPC: As fast as possible**
  **FLOPS/$, FLOPS/W...**
- **Biggest difference: Design effort / FPGA**

⇨ **Enable Tools to make sufficient good trade-offs**

⇨ **Tools rely on suitable input specifications**





## How To Connect…

**What FPGAs can deliver to HPC:**

- **Very fast implementation of static data-paths**
- **Very fast and wide on/off-chip communication (on-chip distributed memories, dedicated links)**
- **2 Approaches**
- **Use FPGA as coprocessor**
  ⇨ **local, late in design process**
- **Consider FPGA at task distribution**
  ⇨ **global, early in design process**

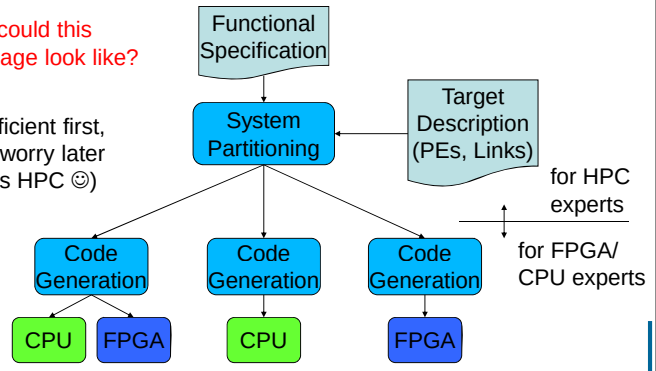




## How To Connect…

How could this language look like?

Be efficient first, don't worry later (this is HPC ☺)



## OUTLOOK





"**HPCwire: You mentioned you're not really interested in FPGAs as accelerators. Why is that?"**
D. Turek (vice president of Deep Computing at IBM):
"[…] I'm not convinced that the software tools and the other things you need for programming them will ever make it, fundamentally."

[http://www.hpcwire.com/hpc/893353.html]





## The Application Mix of the Future

- **Which codes are able to scale to 10000+ processors?**
- **Which codes can benefit from special accelerators?**
- **Who determines if a code is scalable?**
  - The research area,
  - the problem itself,
  - the chosen algorithm or
  - the written code ?
- **What are we going to do with codes that are already hitting scalability limits?**
- **How can we convince people to use better programming languages?**



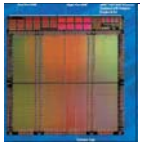


## Accelerators – FPGAs

- **A bright future ahead**
- **Large investments required on the software front**
- **Using this technology usually required advanced knowledge and introduces a large overhead, but the benefits are potentially enormous**
- **More advances relating to double precision issues needed**

## Tomorrow's Hardware?

- **FPGAs and CPUs complement eachother!**
- **Why not combining them on a single chip?**
- **Xilinx (PowerPC) and Altera (ARM) tried, but failed (economically).**
- **Softcores triggered impressive tool development.**
- **With better tools, maybe a second try is due!**



Altera Excalibur EXPA10 die





## Today's Resources

- Is **CERN** a good place to think about how FPGAs could be used for number crunching?
- It certainly features abundant FPGA resources (hidden in LHC DAQ systems)!
- What will they do during LHC **winter shutdown**?









## Acknowledgments

**Fabienne Baud-Lavigne, CERN**
**Francois Flueckiger, CERN**
**Ivica Puljak, Univ. of Split, Croatia**
**Sverre Jarp, CERN openlab**
**Dr. Matthias Brehm, LRZ, Munich, Germany**
**Markus Stürmer, Univ. of Erlangen, Germany**