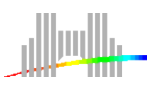
New Architectures, New Compilation Problems

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

April 20, 2010




Introduction

- Due to technology problems, computer architectures are evolving rapidly
- Consequence: new compilation and optimization problems
- Three areas:
  - Power management
  - Reconfigurable architectures (mainly, FPGA)
  - Multicores and manycores

Technology Problems

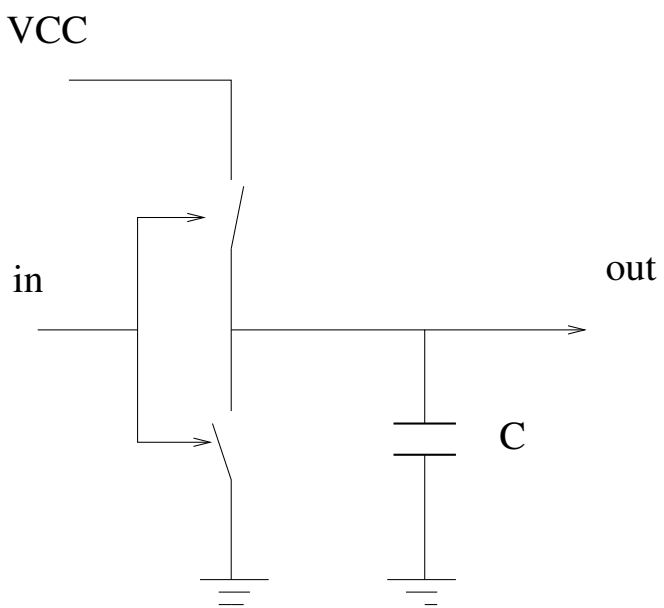
- The feature size is approaching atomic dimensions (32 nm $\approx$ 320 atomic diameters (optimistic evaluation))
- Consequence I: Doubling the density (or dividing the feature size by $\sqrt{2}$) now takes 3 years instead of 18 months
- Consequence II: quantum effects are becoming significant, and generate random errors
- Consequence III: electric current leaks throught thin insulation layers, and creates a static power consumption: the circuit uses power even when doing nothing.

  This is still negligible at 65 nm, but is becoming more important for smaller feature sizes.

Inpact on Architectures and Compilers

- Compilers must become *power aware*
- The hardware in ordinary processors is not always used in full. *Reconfigurable structures* (FPGA) may be more economical
- Excessive power consumption precludes increasing the clock frequency: processing power increase can be found only in *parallelism*.

Where goes the power?

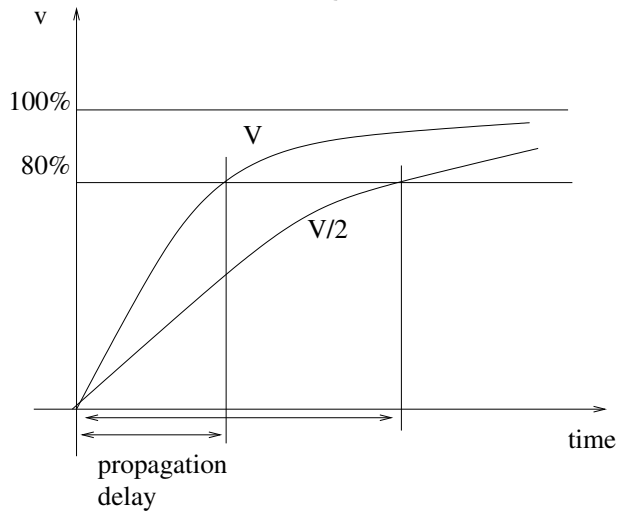


- When connected to the power supply, the capacitor accumulate an energy of the order of $1/2CV^2$
- When switched to the ground, this energy is dissipated in the wiring and comes out as heat
- This happens f times per second (f: the clock frequency), for a proportion $0 < \alpha < 1$ of the gates, for a total dissipation of $1/2\alpha CV^2 f$, where C is the total equivalent capacitance of the chip.

Supply Voltage Reduction

Since power increases like the square of the supply voltage, reduce the voltage, BUT:

- There are physical reasons not to go much below 1 Volt



- Reducing the supply voltage increases the propagation delay, and hence reduces the frequency.

Frequency must be roughly proportional to voltage.

Problems and Solutions

- The power consumption of a fast processor is of the order of 100 W (more than a modern TV set) from a chip size of a few square centimeters
- Large server farms need the output of a small power station
- It is said that serving all Google queries consume a few percents of the Earth energy budget
- For battery powered appliances, reduced power translates into longer battery life.

Reducing power How To:

- Low power technologies
- Parallelism
- Dynamic Power Control
- Improved Algorithms.

Parallelism Reduces Power

- Consider a computation which needs N clock cycles and must be done in time T (e.g. decoding a TV frame)
- The frequency is N/T, the chip equivalent capacity is C, the supply voltage is V, and the energy dissipation is $1/2NCV^2$
- Suppose that the computation can be split in two threads which execute independently on two identical processors
- Each processor execute $N/2$ operations, for a frequency of $N/2T$
- The supply voltage may be halved, the capacity is doubled, and the dissipation $1/2(N/2)2C(V/2)^2$ is divided by 4.

Unfortunately, this kind of perfect parallelism is not frequent. Compilers – especially autoparallelizers – can help.

Dynamic Voltage and Frequency Control

Many chips (including processors) can control their power supply and clock frequency, either step by step or all-or-nothing.
The control may be coarse grain (e.g. per processor in a dual core) or fine grain (ALU, FPU, cache, I/O units ...)

- The compiler can analyze the needs of each phase of a computation, and insert instructions to shut off un-needed components
- Moving like operations around – if possible – increases the effectiveness of this approach
- If given a latency constraint, the compiler may estimate the needed number of cycles, and insert instructions to adjust the clock frequency and supply voltage to their lowest admissible values
- In more complex cases – systems on chip – the frequency of each building block may be adjusted individually.

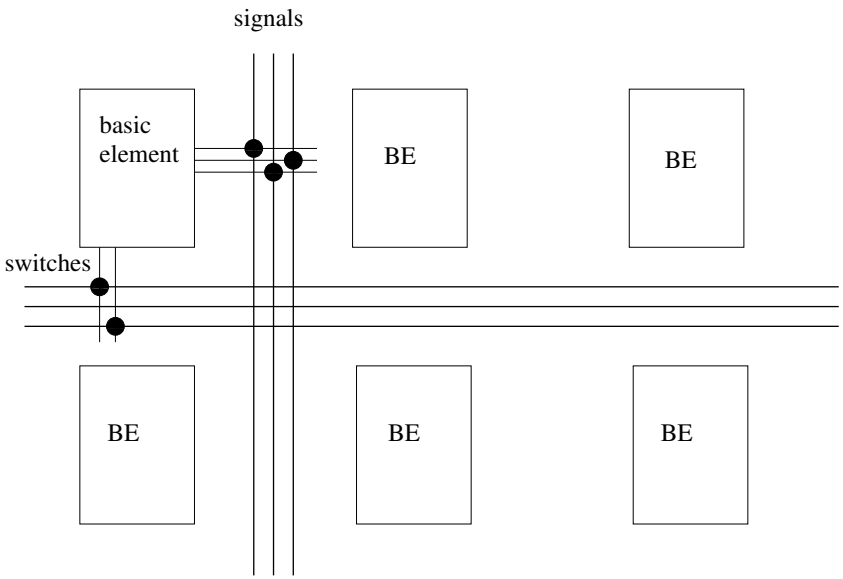
Ordinary Optimizations

Ordinary Optimizations reduce power consumption.

- Locality enhancement, because a cache access uses much less power than a memory access
- Energy is proportional to the number of cycles.
  - constant propagation
  - Instruction level paralellism
  - Algorithm changes

Field Programmable Gate Arrays (FPGA)

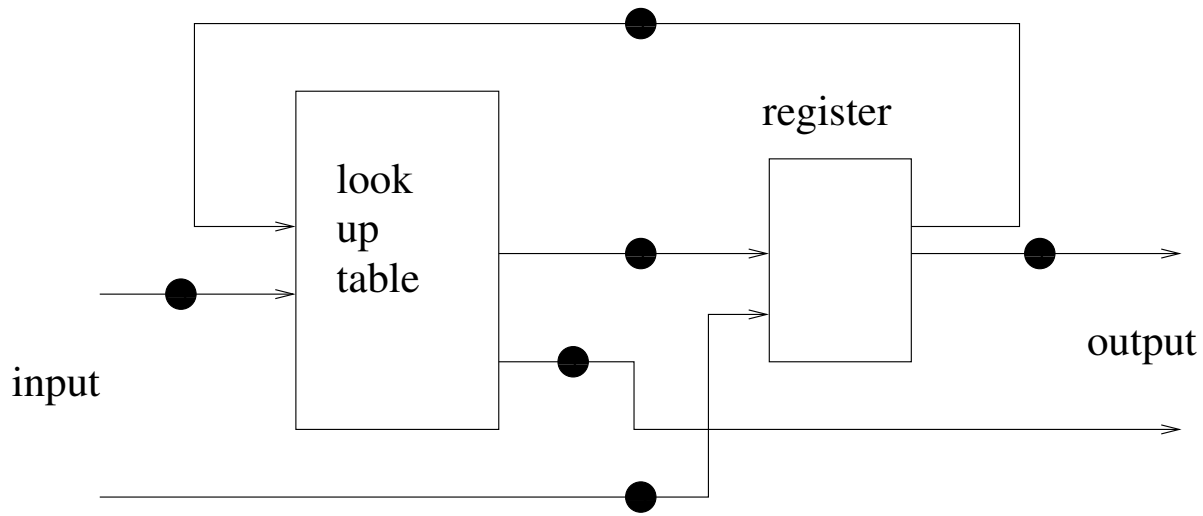
An FPGA is a chip containing a grid of Basic Elements, wires, input/output pins, and a clock network.



Each basic element can be connected to some wires by switches controlled by a configuration layer (memory).

FPGA, II

Each basic element can do arbitrary boolean operations using a look-up table, has some memory, and input and output connections.



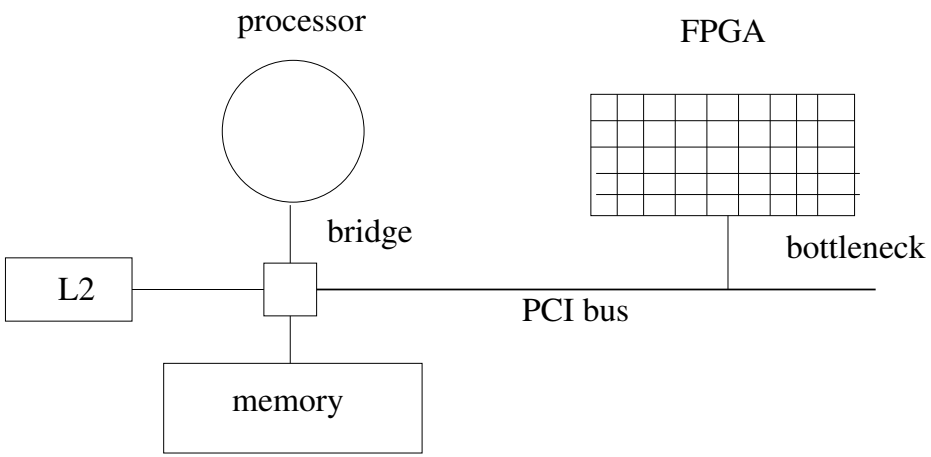
Modern FPGAs may have *predefined blocks*: arithmetic units, memory, and even a complete embedded processor.

Why FPGA?

- They can be reprogrammed simply by changing the position of the switches and the content of the look-up table (unlike Application Specific Integrated Circuits (ASIC) whose operation is fixed at the foundry)
- They are much cheaper than ASIC, at least in term of Design Cost
- In term of performance and power / performance ratio, they are intermediate between ASIC (best) and processors (worst)

Where to use FPGAs?

- As stand alone parts (ASIC replacements) in small markets (radio modems, network processors, routers)
- As accelerators for PC (scientific computing, signal processing, molecular biology)

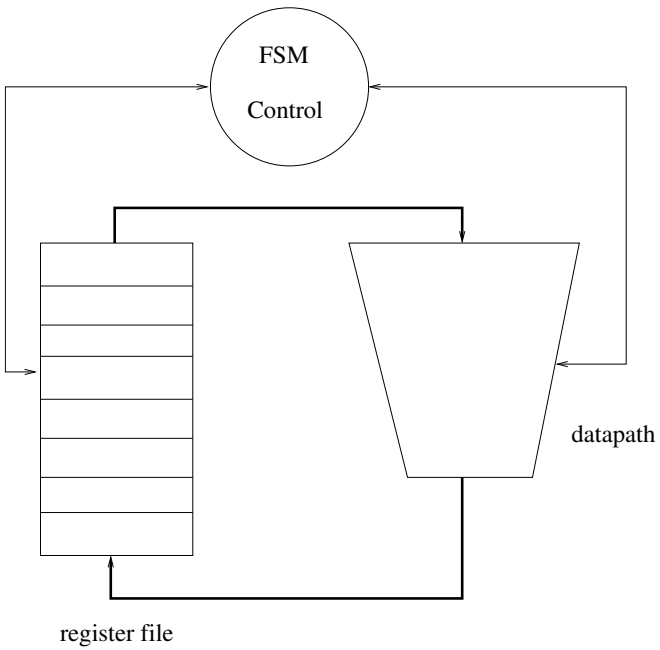


One may imagine that, in a few years, every PC will include an FPGA chip, perhaps with direct access to the main memory. This will happen only if programming an FPGA is as easy as compiling a program.

How to Design an Accelerator

- Identify and extract *kernels* in the application code
  - regular code
  - high processing to memory ratio
- Convert the kernel into a "bit image" for the FPGA: this is High Level Synthesis
- Write the communication code, from the processor to the accelerator and back (the difficult part)

High Level Synthesis



One process

- Convert a behaviour (succession of events in time) ...
- into a structure (set of interconnected components
- Infer the control (Finite State Machine) from the control constructs of the program
- Infer the datapath from the operating statements
- Registers? One register per variable of the program is not the best solution.

Asynchronous Parallelism, I

Build the accelerator as a network of asynchronous communicating processes.

- Parallelism increases performance
- Parallelism decreases power
- Simplify the design of the clock network

Goal: minimize the amount of communications and attempt to implement the "channels" as FirstInFirstOut queues.

- Split the program by statements: 1 statement = 1 process,
- Or try to find outer parallel loops
- No good solution yet.

Why Multicores?

- While power dissipation in silicon can be controlled, power dissipation in the clock tree (metallic strips) cannot, and already accounts for 30 % of the power budget.
- Clock frequency is therefore limited to about 3GHz
- Hidden parallelism (ILP) has found its limits (mainly due to the prevalence of branches in ordinary code)
- Explicit parallelism is necessary, and one of the possibility is to implement several processors on the same chip
- The other solution, VLIW processors (Itanium) seems to have failed.

Facts and Fictions

Dreams

- We soon will have chips with thousands of cores ....
- The number of cores is doubling every eighteen months.

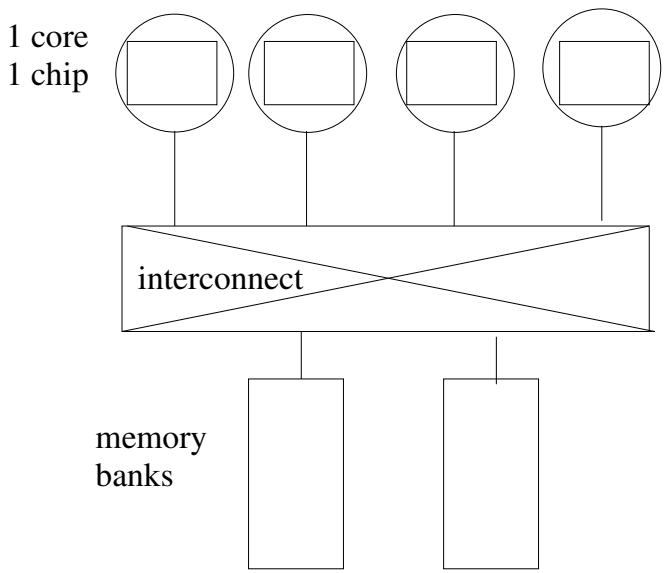
Facts

- Dual-cores are common, quad-cores exist (server applications), six-cores are just coming out (Intel)
- IBM and AMD might be a step ahead
- The Larrabee (80 cores) has been discontinued
- The Cell has only $7 + 1$ cores.

Why?

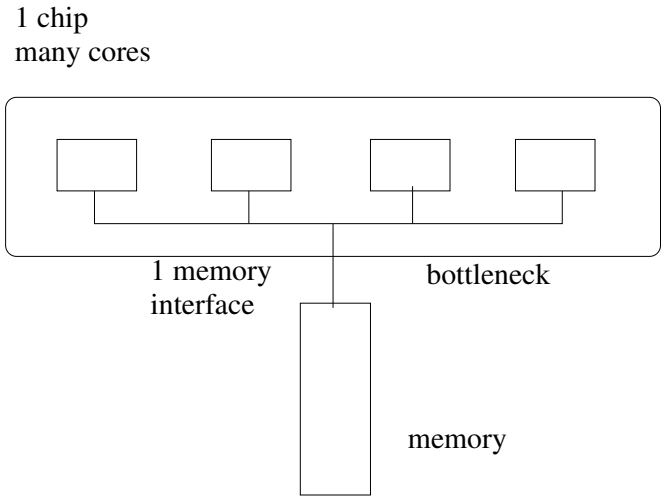
The Memory Wall, I

- Each core needs of the order of one word per cycle
- Including the effect of registers and caches, this translates, **today**, to 20% to 30% of the bandwidth of the memory subsystem



- **Multisockets**: as many memory interfaces as there are sockets, as many memory banks as wanted
- Problem, I: access conflicts to memory banks, hot spots
- Problem II: design of the interconnect.

The Memory Wall, II



- **Multicore**: the bottleneck is the unique memory interface
- Going beyond around 10 cores needs a revolution in memory architecture

The Memory Wall: Solutions

- Increase the size of on-chip memory faster than the core count
- More memory interfaces per chip (IBM has announced a chip with three interfaces)
  - The pin wall
  - Serial memory interfaces ?
- Ask for help from the compiler

What Can the Compiler Do?

Classical Optimizations

- Array shrinking
  - Replace a large array by a smaller one,
  - or even by a scalar,
  - which can be allocated to a register
- Enhance locality
  - Group together in time accesses to the same variable
  - Group together in space related variables (array layouts)
- Program shrinking
  - Remove dead code
  - Propagate constants

Non-Standard Approaches

Consider the on-chip memory as a local memory or scratchpad instead of a cache.

- Most Unix software was designed for very small memories – PDP 11: 64 kbytes, the first PC: 640 kbytes – and fits easilly in the on-chip memory of modern processors
- Some examples (libraries and dynamic data excluded) gcc : 200k, latex: 590k, Open Office 500 k
- Apply the same techniques that are used now for the main memory: bulk loading (not word by word loading) and paging
- May necessitate hardware modifications

What to do for more demanding Applications

High Performance Computing, linear programming and optimization, database search, ...

- Include the on-chip memory management into the application (with help from the Operating System) – a nice application for aspect oriented or feature oriented programming
- Have the compiler generate the memory management code

A Word about GPUs

GPUs are manycores, each core being a vector processor, plus specialized units for graphic processing (sines and cosines for rotations, texture ...)
How is the memory wall problem solved?

- GPUs use streaming memories, which have much lower latencies than ordinary memories, *provided successive accesses are at consecutive addresses*
- Streaming memories were created for fast refresh of a display, and are well adapted for vector processing
- It seems likely that part of the GPU technology will percolate into general purpose processors.

A Word in Conclusion

As always, computer architecture is changing faster than compilers can follow.

Compilers are enabling technologies for some recents evolutions:

- Autoparallelization for multicores
- High Level Synthesis for FPGAs
- Power management for battery driven appliances