Faculté des Sciences et Ingénierie
Master Informatique
Systèmes Électroniques, Systèmes Informatiques

PROGPAR

Courses

Adrien Bourmault
(adrien.bourmault@etu.upmc.fr)

Enseignant : Adrien Cassagne

Table des matières

Chapitre 1

Introduction

This class focuses on NVIDIA Jetson TX2, an embedded architecture from the same family than the Nitendo Switch. It is used in cars, satellites, etc.

This isn't a toy but a devkit.

Chapitre 2

Programmable architectures

2.1 Simplified CPU architecture

What we do in CS is using instructions to manipulate memory. More precisely, there are :
— control instructions
— arithmetic instructions
— memory instructions

2.2 CPU Architecture

On modern CPU, CPU/RAM freq is between 1GHz and 4GHz
Memory throughput is 50 GB/s (DDR5)
But CPU is much faster (Œ100), which is solved by caching.

2.2.1 Memory hierarchy

Most applications often reuse the same data.
We have :
— L1 : 2 cycles
— L2 : 10 cycles
— L3 : 30 cycles
Storing implies two methods :
— writethrough : write to both RAM and cache, a choice made in some architectures, leading to complex design in CPU arch.
— writeback

2.2.2 spatial locality

While one cache line is a certain amount of data and the smallest packet between RAM and CPU is a cache line, it may be interesting to access data contiguously

2.2.3 SIMD

With scalar instructions, in 1 cycle, with two operands you create the content of a register.

With SIMD, in 1 cycle you do that with 4 operations at the same time. The registers are larger than regular ones, and called differently : vector registers.

2.3 Co-processors

Hardware separated from CPU, connected often via PCIe, with its own RAM.

2.4 GPU Architecture

Generally much more parallel than CPUs, fewer control parts than CPUs but much more compute units.

Memory is faster : 500 GB/s

It is often suitable for scientific computing but not always !

Hardware acceleration for AI (tensor cores) are often present. Not studied in this course.

2.4.1 Nvidia Ampere

We can see there are a lot of cores. Much more than in a CPU.

2.5 Supercomputer architecture

Here we design that as 8 computers connected via a switch. Theorical max performance is 8 times that of a node.

Chapitre 3

Single-core CPU architecture

3.1 Pipeless processors

[Remembering about pipeline]

3.2 Pipelined processors

[Remembering about pipeline]
Note : pipeline makes latency higher
Modern CPUs are 10 to 20 stages pipelined.
Conditional branchment implies that it's difficult to guess which instruction is the next, which is bad for pipeline efficiency (it created bubbles) so we have to predict... with branch prediction, to limit the effect.

3.3 Superscalar processors

[Remembering about this]
ILP = superscalar CPU fetcher/decoder

3.4 Out of order execution (OoO)

The CPU can reorder instructions basing on dependencies of instructions.

3.5 Example : Apple Silicon M1

This is an OoO CPU, since there is a dispatcher that has a reorder buffer.
There is a big part dedicated to SIMD.

3.6 Example : Intel Alder Lake

Much larger decoding part (yeah, CISC).

3.7 So... our new toy ?

There are :
— 2 Œ powerful Nvidia Denver 2 cores @ 2.04 GHz

— 4 Œ powerful ARM Cortex A57 (Big) cores @ 2.0GHz
— Weird heterogeneous design
— No energy efficient cores

3.7.1 Cortex A57 (2015)

This is a 18-stage pipelined CPU, with 3-way decoder and it's 8-wide super-scalar. This is a OoO CPU.

3.7.2 Nvidia Denver 2 (2016)

Not much info about it...
We have :
— 15-stage pipeline
— 2-way decoder
— 7-wider superscalar
— in-order execution (ARM code translated to something by a hardware translator, so it can be considered OoO since that translator reorders !)

Side note : the project was to decode ARM and x86 assembly on the same CPU, but Intel didn't gave the license

3.8 Nvidia Jetson TX2 topology

[See slides]

Chapitre 4

Single-core CPU optimizations

It's a good idea to rely on compilers for the sake of readability.
Compilers provide optimizations with '-O' options.
There is '-march=native' that is sometimes the only way to allow code vectorization.

4.1 Latency and throughput

We have :
— add : 4 cycles, 0.5 CPI
— sub : idem
— mul : idem
— div : 14 cycles, 4 CPI
For the 3 first, we can have 2 inst/cycle !

4.1.1 Example : division

Here we see than we should avoid divisions (and use preprocessor to rewrite that as something else).

4.2 Special functions

On modern CPUs we have some common mathematical functions. These are often expensive, except for rsqrt (used for 3D/2D calculation)

4.3 Function calls

Sometimes it's better to avoid calls, especially depending on where the function call is located, for example with the stencil algorithm.
Inlining is a good idea.

4.4 Loop unrolling

It reduces the time spent in loop control, and especially the risk of branch prediction error.
In fact, it also increases optimization opportunities while exposing more parallelism for ILP, masks instruction latency.