Parallel Systems

Project topics 2016 - 2017

1. Scheduling

Scheduling is a common problem which however is NP-complete, so that we are never sure about the optimality of the solution. Parallelisation is a natural way to find better solutions. The way the search space is traversed will be discussed in the forthcoming chapter on Discrete Optimization Problems (slides can be found on website).

Parallelisation can happen on several levels (see slide). We will provide a real problem and discuss at which level the student will parallelize the code.

Automated warehouses: scheduling

Optimization of container barge routing

scheduling

Parallelization

Algorithms

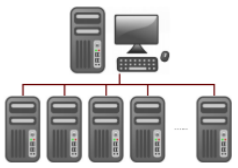
- Branch & Bound
- Iterated Local Search
- …

Levels of Parallelism

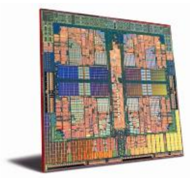
- Alternative algorithms
- Alternative scenarios (due to uncertainty)
- Multi-search
- Tree-level parallelism
- Node-level parallelism (neighbor exploration)
- parallel calcution of score or bound

Parallel Hardware

- cluster



- multicore



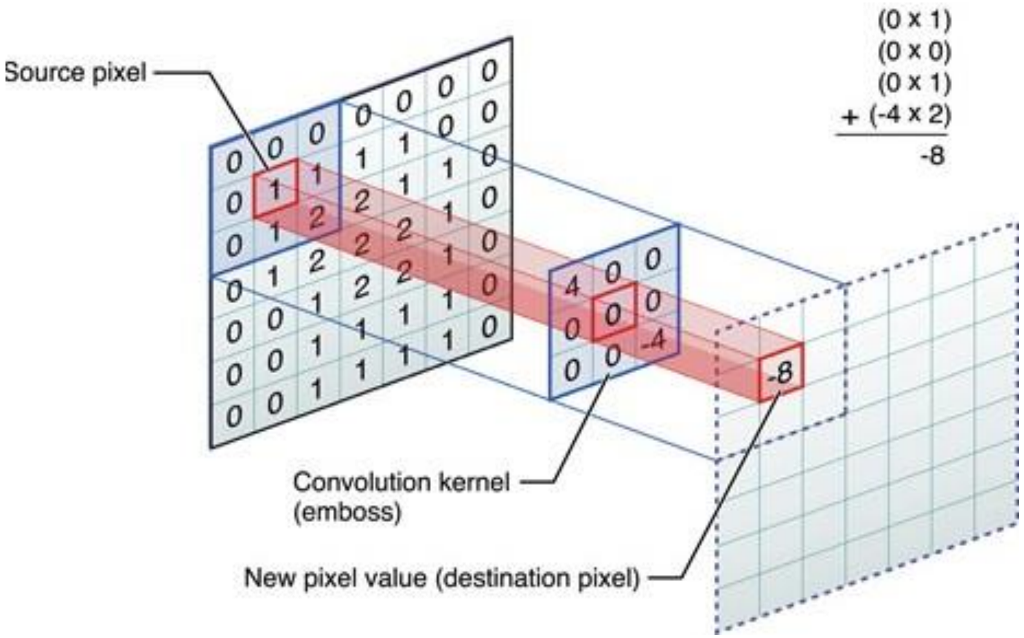
- accelerator

Suitability is problem-dependent!

2. Image processing

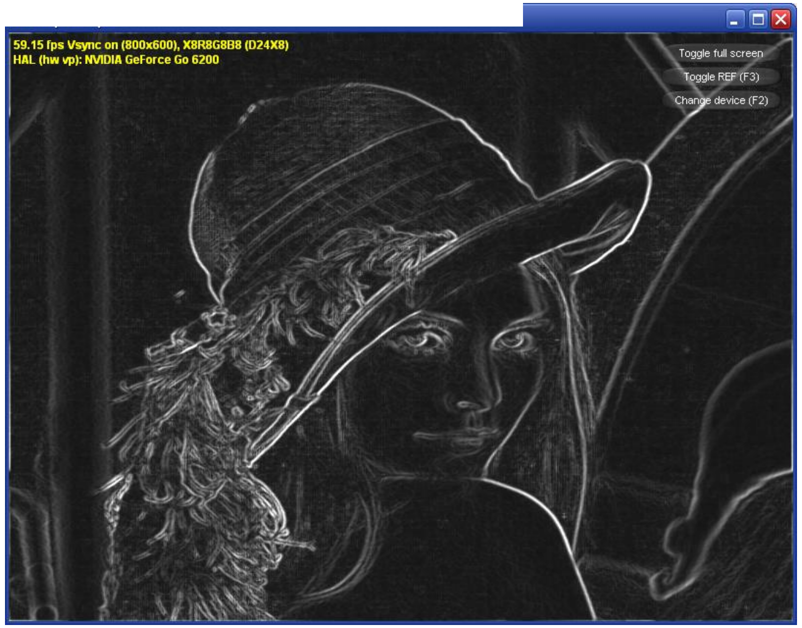
Image processing requires a lot of computational power (and also memory bandwidth). These algorithms are therefore first choices for parallelizations.

- Burg – chapter 3 – convolutions
  - Neighborhood operation



Examples of convolution

Edge detection
with sobel filter



Histogram

- Histogram calculations are essential parts of many algorithms. It is not completely straight forward to do this on GPU, but with some ‘tricks’, a good speedup can be achieved.
- Company On Semi is interested in a GPU implementation to calibrate and test the image sensors they are fabricating.

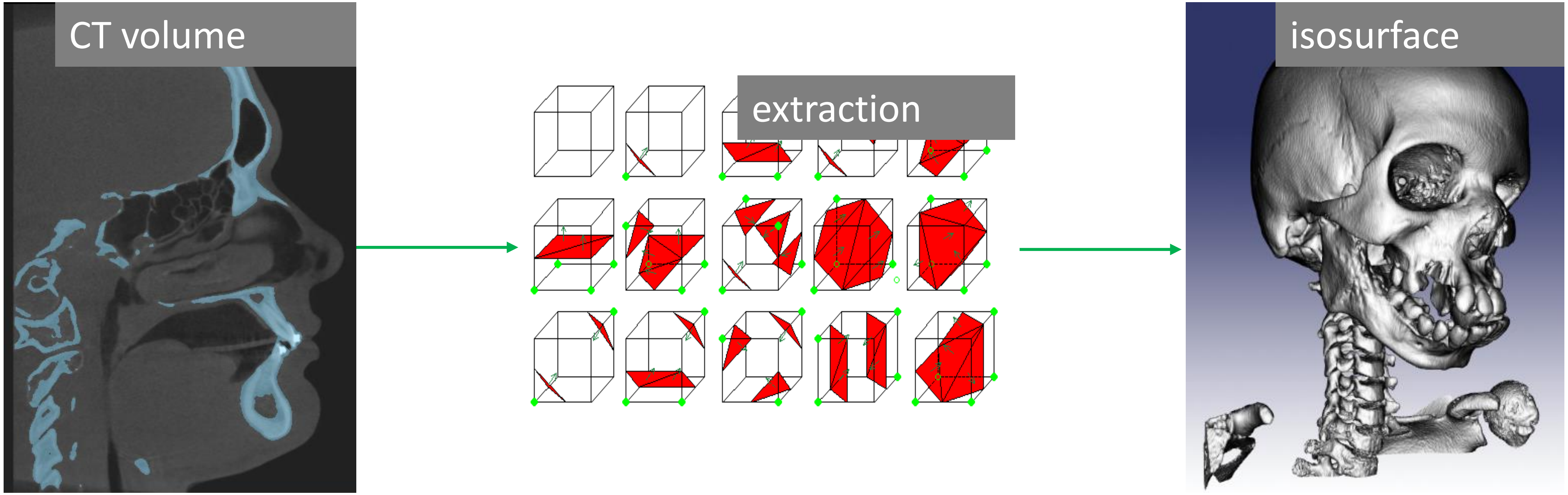
Image processing for dentists

Nobel Biocare develop software for dentists and surgeons. They need to speed up their code to ensure a nice user experience. At the moment, they are stuck with algorithms based on **convolutions** (see earlier slides). Convolutions are perfect for GPUs!

The following 2 slides show previous things we did with Nobel Biocare.

5.6 Marching cubes

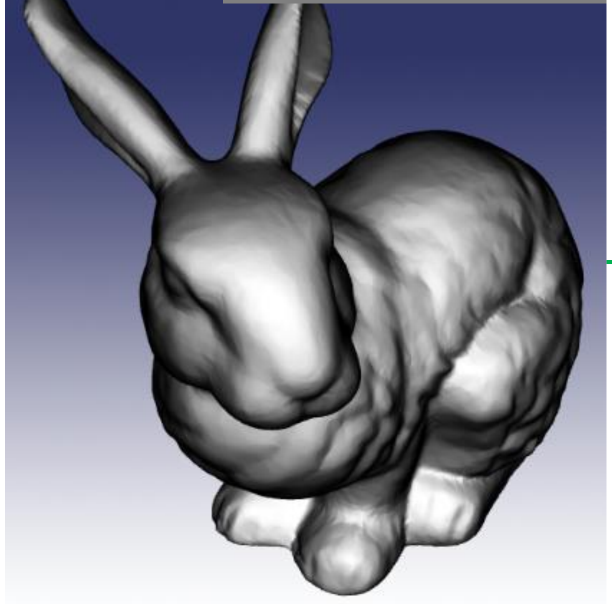
Requirements:
- Speed up algorithm (MACH: seconds -> real-time).
- Heterogenous CPU/GPU portability (Win/Mac).



5.6 Distance map (distance field)

Requirements:
- Speed up algorithm (MACH: minutes -> seconds).
- Heterogenous CPU/GPU portability (Win/Mac).

3D model



ray vs. triangle

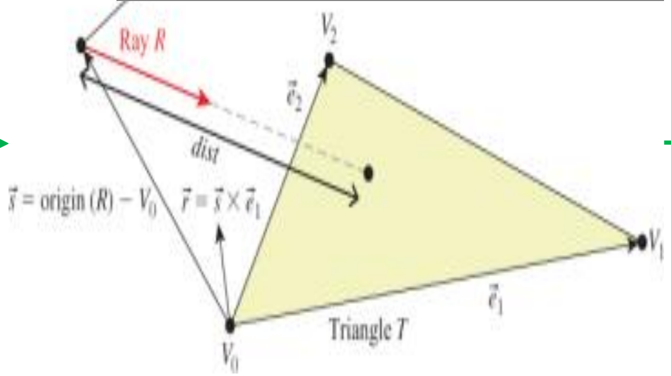
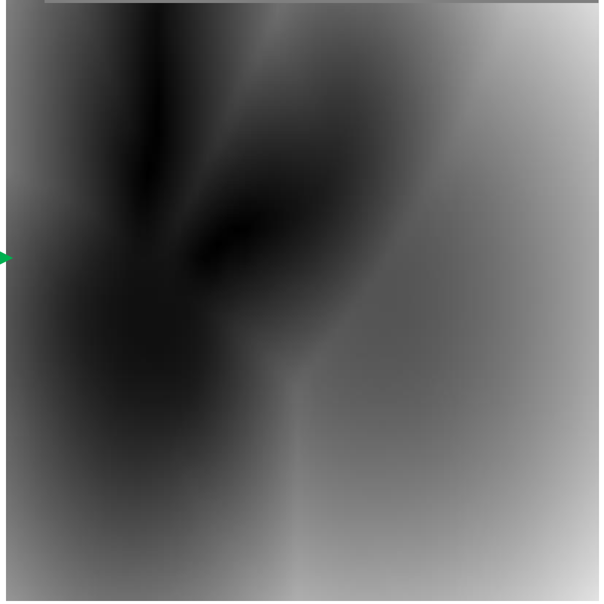


image volume



The fastest determinant calculator in the world

Calculating determinants take time. A student of last year successfully build a GPU implementation (and also a python version). Calculating a determinant is based on the determinants of sub matrices. His algorithm starts with calculating all determinants of all 2x2 sub matrices, then 3x3 and so on. It works!

There are still some optimizations to be tested. We want to make it available for being used in Matlab.

An application is security. They are looking for matrices (see next slide) for which no determinant of any sub matrix is zero. Our GPU implementation will help, because their matlab code is awfully slow!

4-dimensional matrix

$$[A_{4\times4}] = \begin{bmatrix} 1 & a_{12} & 0 & 0 \\ b_{12} & 1+a_{12}b_{12} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & a_{13} & 0 \\ 0 & 1 & 0 & 0 \\ b_{13} & 0 & 1+a_{13}b_{13} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & a_{14} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ b_{14} & 0 & 0 & 1+a_{14}b_{14} \end{bmatrix} \times$$

$$\times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & a_{23} & 0 \\ 0 & b_{23} & 1+a_{23}b_{23} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & 1 & 0 \\ 0 & b_{24} & 0 & 1+a_{24}b_{24} \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a_{34} \\ 0 & 0 & b_{34} & 1+a_{34}b_{34} \end{bmatrix}$$

where $a_{ij} = a, b_{ij} = b,$ *we have*

$$[A_{4\times4}] = \begin{bmatrix} 1 & a & 0 & 0 \\ b & 1+ab & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ b & 0 & 1+ab & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ b & 0 & 0 & 1+ab \end{bmatrix} \times$$

$$\times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & a & 0 \\ 0 & b & 1+ab & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & a \\ 0 & 0 & 1 & 0 \\ 0 & b & 0 & 1+ab \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & a \\ 0 & 0 & b & 1+ab \end{bmatrix}$$

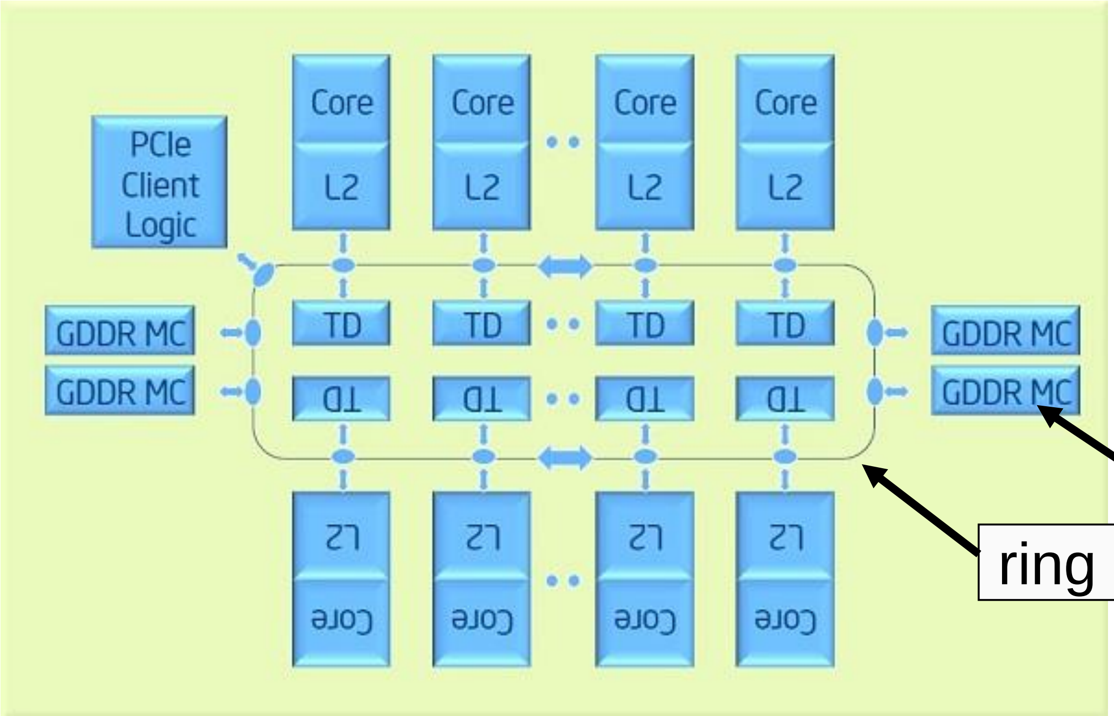
$$= \begin{bmatrix} 1 & a & a & 0 \\ b & 1+ab & ab & 0 \\ b & 0 & 1+ab & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & a & 0 \\ 0 & b & 1+ab & 0 \\ b & 0 & 0 & 1+ab \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & ab & 0 \\ 0 & 0 & 1 & a \\ 0 & b & b(1+ab) & (1+ab)(1+ab) \end{bmatrix}$$

Test the Intel Xeon Phi

A few years ago, Intel brought the Xeon Phi coprocessor on the market to compete with the huge processing power of GPUs. We have one at our department. A student could test it capabilities and compare it with GPUs.

The main difference with GPUs is that you need to vectorize your code. This can be done manually or by compiler pragmas (see forthcoming slides).

Intel’s Xeon Phi coprocessor



Intel’s response to GPUs…

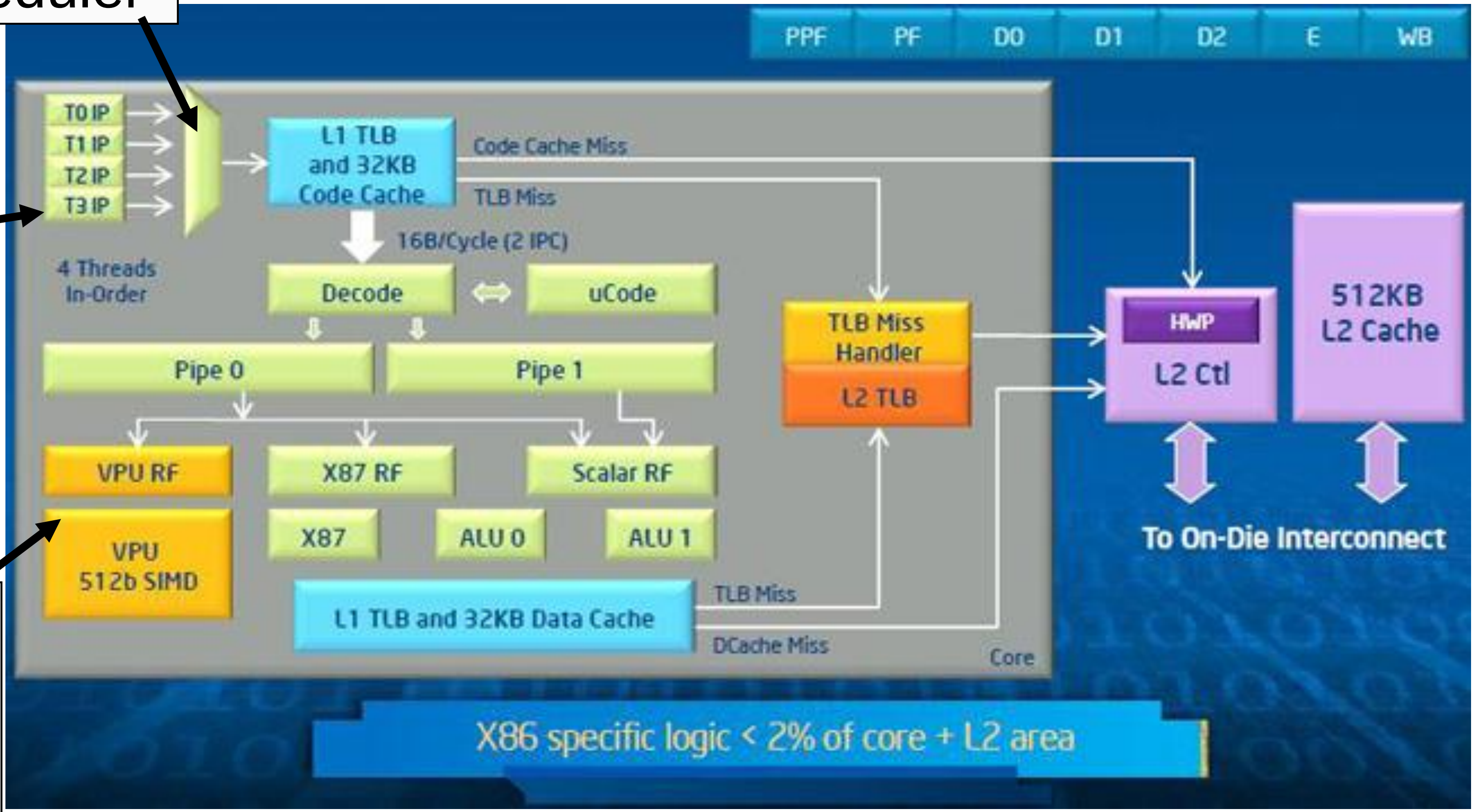
60 cores

Intel's Xeon Phi's core

Thread scheduler

4 hardware threads

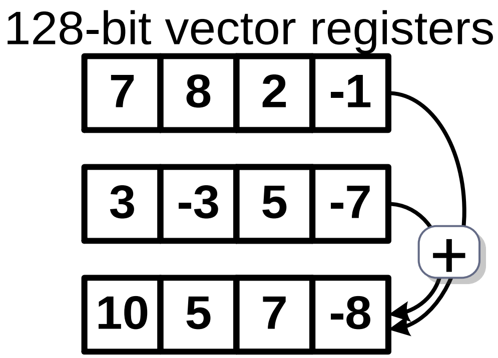
512-bit Vector unit (SIMD)



Usage of the coprocessor

- As MPI-node
- Off-load from main processor
- As standalone processor
- Common c-programming
  - Pthreads
  - Openmp
  - Intel threading building blocks

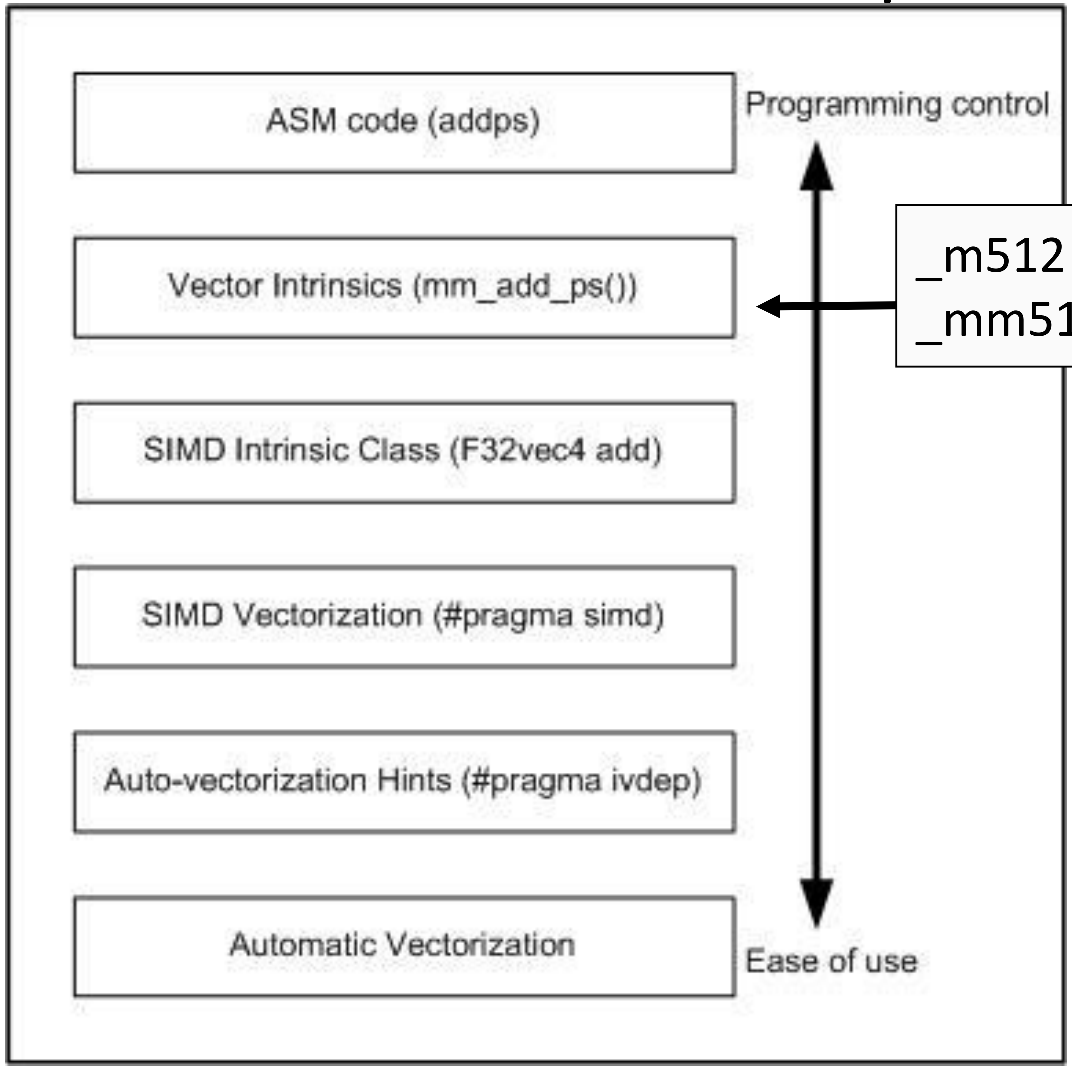
Vector processors (SIMD)



Instructions can be performed at once on all elements of vector registers

- Has long be viewed as the solution for high-performance computing
  - Why always repeating the same instructions (on different data)? => just apply the instruction immediately on all data
- However: difficult to program
- Is SIMT (OpenCL) a better alternative??

Vectorization needed for peak performance



_m512 register
_mm512_add operations

Auto-vectorization

```
roewaele@knc-2:~/Projects/adhd/simpleflops$ CFLAGS="-vec-report6 -mmic" CC=icc make -B
icc -vec-report6 -mmic -std=c99 -fopenmp -O3    test.c   -lrt -o test
test.c(36): (col. 4) remark: loop was not vectorized: existence of vector dependence.
test.c(36): (col. 4) remark: vector dependence: assumed FLOW dependence between  line 36 and  line 3
test.c(36): (col. 4) remark: vector dependence: assumed ANTI dependence between  line 36 and  line 3
test.c(36): (col. 4) remark: vector dependence: assumed FLOW dependence between  line 36 and  line 3
test.c(36): (col. 4) remark: vector dependence: assumed ANTI dependence between  line 36 and  line 3
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(36): (col. 4) remark: vector dependence: assumed OUTPUT dependence between a.256 line 36 and
test.c(30): (col. 2) remark: loop was not vectorized: not inner loop.
test.c(9): (col. 2) remark: loop was not vectorized: existence of vector dependence
```

SIMD pragma to indicate parallism

```
void dflops(double * restrict a) {
  const double c = 1.;
  const double x = 0.9;
#pragma simd
  for (long long i = 0; i < niterations; i += 16) {
    a[0] = a[0] * x + c;
    a[1] = a[1] * x + c;
    a[2] = a[2] * x + c;
    a[3] = a[3] * x + c;
    a[4] = a[4] * x + c;
    a[5] = a[5] * x + c;
    a[6] = a[6] * x + c;
    a[7] = a[7] * x + c;

    a[8] = a[8] * x + c;
    a[9] = a[9] * x + c;
    a[10] = a[10] * x + c;
    a[11] = a[11] * x + c;
    a[12] = a[12] * x + c;
    a[13] = a[13] * x + c;
    a[14] = a[14] * x + c;
    a[15] = a[15] * x + c;
  }
}
```

Successful vectorization

```
rdewaele@knc-2:~/Projects/adhd/simpleflops/simpleflops$ CFLAGS="-vec-report6 -mmic"
icc -vec-report6 -mmic -std=c99 -O3 -fopenmp -funroll-loops -vec-report=6    test.c
test.c(84): (col. 5) remark: vectorization support: reference sa has unaligned acce
test.c(84): (col. 5) remark: vectorization support: unaligned access used inside lo
test.c(83): (col. 4) remark: LOOP WAS VECTORIZED.
test.c(76): (col. 3) remark: loop was not vectorized: not inner loop.
test.c(74): (col. 2) remark: loop was not vectorized: not inner loop.
test.c(79): (col. 4) remark: SIMD LOOP WAS VECTORIZED.
test.c(13): (col. 2) remark: SIMD LOOP WAS VECTORIZED.
test.c(38): (col. 2) remark: SIMD LOOP WAS VECTORIZED.
```

Test new features of OpenCL 2.0

Explore new features

- You need an OpenCL 2.0-enabled GPU (we can buy one)
- Check online
  - Explore differences with OpenCL1.0
- Test them, play with them: make small programs that show how they work
- Features
  - Memory mapping (one address space CPU-GPU)
  - Streams
  - ...

Microbenchmarks

Our GPU research consists of developing small programs that test a specific feature of GPUs (e.g. memory bandwidth, double precision performance, overhead time for launching a kernel, etc). They are called microbenchmarks and can be found on: www.gpuperformance.org.

We invite students to develop new microbenchmarks that we can add to our microbenchmark suite.