GPU research in the ES-group

Henk Corporaal (professor)
Gert-Jan van den Braak (postdoc)
Roel Jordans (postdoc)
Erkan Diken (PhD)
Rik Jongerius (PhD)
Ang Li (PhD)
Maurice Peemen (PhD)
Luc Waeijen (PhD)
Mark Wijtvliet (PhD)
Welcome to PARsE
Welcome to the website of the parallel architecture research team. The PARsE team is a subdivision of the Electronic Systems group, part of the Electrical Engineering department at Eindhoven University of Technology (TU/e) in The Netherlands.

Contents of the website
This website contains information on previous, current and future research, education and events. We update the research log daily with new posts concerning the latest updates in research, discussing articles published at journals and conferences from among others ACM and IEEE. You will find the following sections on the website:

- A daily updated research log.
- An overview of our research, including projects, tools and publications.
- The education page, containing course and student project information.
- Events, such as our GPU symposium.
- More information about us.

Latest research log entries
Microserver has landed
The first microserver from IBM/Astron has arrived at the TU/e. We have started benchmarking the microserver. Next, we plan to update our Bones source-to-source compiler and add the microserver as a target in the coming weeks.
PARsE
Parallel Architecture Research Eindhoven

• Using advanced heterogeneous platforms
  • Multi-core CPUs
  • GPUs
  • DSPs
  • FPGAs

• Efficient code generation
  • Code transformation & generation
  • Compilers

• Even more efficient: new architectures
  • SIMD, CGRA, R-GPU
  • Accelerators
    – Neural networks (CNNs)
GPU research – overview (selection)

- Application mapping
  - Histogram, CNN

- Understanding GPUs
  - Modeling of GPU L1 cache
  - Cache bypassing

- Architecture modification
  - Hash functions in scratchpad memory

- Code generation
  - Bones source-to-source tools
Application mapping

• Histogram,

• Convolutional Neural Networks (CNN)
Application mapping: histogram

- Load pixel
- Update votes

Histogram – replication

- Load pixel
- Update votes

Scratchpad memory layout

- Scratchpad memory
  - Divided in 32 banks
  - Each bank has 32 lock-bits, 1024 in total
Application mapping: CNN

- Convolutional Neural Network (CNN)
  - GTX 460: 35fps
  - Tegra X1: ~20fps

Layer 1: 6\times358\times638
6x6 conv. with 2x2 subsample

Layer 2: 16\times177\times317
6x6 conv. with 2x2 subsample

Layer 3: 80\times173\times313
5x5 conv.

Layer 4: 8\times173\times313
1x1 conv.

Object Category + Position at (x, y)

[4] Speed Sign Detection and Recognition by Convolutional Neural Networks
Understanding GPUs

- Modeling of GPU L1 cache
- Cache bypassing
- Transit model
Understanding GPUs: L1 cache modeling

- GPU Cache model:
  - Execution model (threads, thread blocks)
  - Memory latencies
  - MSHRs (pending memory requests)
  - Cache associativity

[5] A Detailed GPU Cache Model Based on Reuse Distance Theory
L1 cache model – results

Mean absolute error of 6.4%
Understanding GPUs: Cache bypassing

- Cache Insensitive (CI)
- Moderate Cache Sensitive (MCS)
- Highly Cache Sensitive (HCS)

Memory System Throughput vs. Thread Volume

Vertical Design (Bypass based on Operations)
- op0 bypass;
- op1 cache;
- op2 cache;
- op3 bypass;
- op4 cache;
- op5 bypass;
- ...

Horizontal Design (Bypass based on Warp Index)
- All Cache:
  - op0;
  - op1;
  - op2;
  - op3;
  - op4;
  - ...
- All Bypass:
  - op0;
  - op1;
  - op2;
  - op3;
  - op4;
  - ...

Cache bypassing – results

[Image of a bar chart and a histogram showing execution time and normalized IPC for different benchmarks under different cache bypassing techniques.]
Understanding GPUs: Transit model

- Transit model: computation and memory sub-systems

Architecture modifications

- Scratchpad memory hash functions
- R-GPU

Hash function

0000 0001 0001 00

lock bank
GPU modifications: bank & lock conflicts

addr = 32 * id

address
0000 00001 00000 00
lock  bank

all addresses in bank 0
Resolving bank conflicts: hash functions

\[ \text{addr} = 32 \times \text{id} \]

Hash function:

\[
\begin{array}{cccc}
0000 & 00001 & 00000 & 00
\end{array}
\]

lock
bank
Resolving bank conflicts: hash functions

addr = 33 * id

Hash function

0000 00001 00001 00

lock  bank

Configurable XOR Hash Functions for Banked Scratchpad Memories in GPUs
Architecture modifications: R-GPU
How to generate efficient code for all these devices?
Code generation: ASET & Bones

```
Algorithmic Species Extraction Tool

sequential C code

species-annotated C code

skeleton-based compiler

PET (llvm)

Multi-GPU (CUDA / OpenCL)
CPU-OpenMP
GPU-OpenCL-AMD
CPU-OpenCL-Intel XeonPhi-OpenCL
GPU-CUDA
FPGA

\[10\] Automatic Skeleton-Based Compilation through Integration with an Algorithm Classification
```
Example C to CUDA transformation

Example 1: Sum

```c
int sum = 0;
for (int i=0; i<N; i++){
    sum = sum + in[i];
}
```

```c
template <unsigned int blockSize>
__device__ void warpReduce(volatile int *sm, unsigned int tid) {
    if (blockSize >= 64) sm[tid] += sm[tid + 32];
    if (blockSize >= 32) sm[tid] += sm[tid + 16];
    if (blockSize >= 16) sm[tid] += sm[tid + 8];
    if (blockSize >= 8) sm[tid] += sm[tid + 4];
    if (blockSize >= 4) sm[tid] += sm[tid + 2];
    if (blockSize >= 2) sm[tid] += sm[tid + 1];
}
```

```c
template <unsigned int blockSize>
__global__ void reduce6(int *g_idata, int *g_odata, unsigned int n) {
    extern __shared__ int sm[];
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(blockSize*2) + tid;
    unsigned int gridSize = blockSize*2*gridDim.x;
    sm[tid] = 0;
    while (i < n) {
        sm[tid] += g_idata[i];
        sm[tid] += g_idata[i+blockSize];
        i += gridSize;
    }
    __syncthreads();
    if (blockSize >= 512) {
        if (tid < 256) { sm[tid] += sm[tid + 256]; } __syncthreads();
    }
    if (blockSize >= 256) {
        if (tid < 128) { sm[tid] += sm[tid + 128]; } __syncthreads();
    }
    if (blockSize >= 128) {
        if (tid <  64) { sm[tid] += sm[tid +  64]; } __syncthreads();
    }
    if (tid < 32) { warpReduce<blockSize>(sm, tid); }
    if (tid == 0) { g_odata[blockIdx.x] = sm[0]; }
}
```
Second example: maximum

Example 1: Sum

```cpp
int sum = 0;
for (int i=0; i<N; i++){
    sum = sum + in[i];
}
```

Example 2: Max

```cpp
int max = 0;
for (int i=0; i<N; i++){
    max = (max>in[i])?max:in[i];
}
```

template <unsigned int blockSize>
    __device__ void warpReduce(volatile int *sm, unsigned int tid) {
        if (blockSize >= 64) sm[tid] = (sm[tid]>sm[tid+32]) ? sm[tid] : sm[tid+32];
        if (blockSize >= 16) sm[tid] = (sm[tid]>sm[tid+ 8]) ? sm[tid] : sm[tid+ 8];
        if (blockSize >=  8) sm[tid] = (sm[tid]>sm[tid+ 4]) ? sm[tid] : sm[tid+ 4];
        if (blockSize >=  4) sm[tid] = (sm[tid]>sm[tid+ 2]) ? sm[tid] : sm[tid+ 2];
        if (blockSize >=  2) sm[tid] = (sm[tid]>sm[tid+ 1]) ? sm[tid] : sm[tid+ 1];
    }

template <unsigned int blockSize>
    __global__ void reduce6(int *g_idata, int *g_odata, unsigned int n) {
        extern __shared__ int sm[];
        unsigned int tid = threadIdx.x;
        unsigned int i = blockIdx.x*(blockSize*2) + tid;
        unsigned int gridSize = blockSize*2*gridDim.x;
        sm[tid] = 0;
        while (i < n) {
            sm[tid] = (sm[tid]>g_idata[i]) ? sm[tid] : g_idata[i];
            sm[tid] = (sm[tid]>g_idata[i+blockSize]) ? sm[tid] : g_idata[i+blockSize];
            i += gridSize;
        }
        __syncthreads();
        if (blockSize >= 512) {
            if (tid < 256) { sm[tid] = (sm[tid]>sm[tid+256]) ? sm[tid] : sm[tid+256]; }
            __syncthreads();
        }
        if (blockSize >= 256) {
            if (tid < 128) { sm[tid] = (sm[tid]>sm[tid+128]) ? sm[tid] : sm[tid+128]; }
            __syncthreads();
        }
        if (blockSize >= 128) {
            if (tid <  64) { sm[tid] = (sm[tid]>sm[tid+ 64]) ? sm[tid] : sm[tid+ 64]; }
            __syncthreads();
        }
        if (tid < 32) { warpReduce<blockSize>(sm, tid); }
        if (tid == 0) { g_odata[blockIdx.x] = sm[0]; }
    }
**Matrix-vector multiplication:**

\[
\begin{align*}
0:63,0:127 &| \text{chunk}(0:0,0:127) \uparrow 0:127 | \text{full} \rightarrow 0:63 | \text{element} \\
&
\begin{aligned}
\text{for (i=0; i<64; i++)} & \{ \\
\quad r[i] &= 0; \\
\quad \text{for (j=0; j<128; j++)} & \{ \\
\quad \quad r[i] &= r[i] + M[i][j] * v[j]; \\
\quad \} \\
\} \\
\end{aligned}
\end{align*}
\]

**Stencil computation:**

\[
\begin{align*}
1:126 | \text{neighbourhood}(-1:1) \rightarrow 1:126 | \text{element} \\
&
\begin{aligned}
\text{for (i=1; i<128-1; i++)} & \{ \\
\quad m[i] &= 0.33 \times (a[i-1] + a[i] + a[i+1]); \\
\} \\
\end{aligned}
\end{align*}
\]
What do we gain in performance?

[12] Bones: An Automatic Skeleton-Based C-to-CUDA Compiler for GPUs
Education – mapping assignments

• Master course
  • contrast enhancement
  • Viola-Jones Face Detection
  • SIFT object recognition
  • convolutional neural network (CNN)
  • ‘bitcoin’ mining

• Post-master (PDEng) course:
  • GPU & cluster computing
Student projects

• Accelerating AURORA on Multi-Core and Many-Core Processor Architectures – VITO, Belgium

• Advanced ultrasound beam forming using GPGPU technology – esaote, Maastricht

• Domain Transform Acceleration for the GPU-Based Real-Time Planar Near-Field Acoustic Holography

• Analysis and Modeling of the Timing Behavior of GPU Architectures, TU/e
Summary

- Research topics:
  - Application mapping
  - Understanding GPUs
  - Architecture modifications
  - Code generations

- MSc. students, PDEngs & PhDs

- More on the website:
  - http://parse.ele.tue.nl/