Platform and Research overview on the Intel Single-chip Cloud Computer

Invited talk at EPITA, Paris

Roy Bakker
R.Bakker@uva.nl

Computer Systems Architecture
Informatics Institute
Faculty of Science
University of Amsterdam

October 17, 2012
Outline

1. About me
2. The SCC
3. Power Management on the SCC
4. ManyMan - Multi-touch Multicore Manager
5. Current Work: Process Networks on the SCC
6. SVP on the SCC
Outline

1. About me
2. The SCC
3. Power Management on the SCC
4. ManyMan - Multi-touch Multicore Manager
5. Current Work: Process Networks on the SCC
6. SVP on the SCC
About me

- Born
  1987
About me

- Born
  1987
- Highschool Den Burg Texel
  1999-2005
About me

- Born
  1987
- Highschool Den Burg Texel
  1999-2005
- BSc Computer Science UvA
  2005-2008

Roy Bakker (UvA - CSA)
The Intel SCC
October 17, 2012
About me

- Born 1987
- Highschool Den Burg Texel 1999-2005
- BSc Computer Science UvA 2005-2008
- MSc Computer Science UvA 2008-2011
About me

- Born 1987
- Highschool Den Burg Texel 1999-2005
- BSc Computer Science UvA 2005-2008
- MSc Computer Science UvA 2008-2011
- PhD Computer Architecture 2011-(2015?)
Outline

1. About me

2. The SCC
   - The Architecture
   - Some measurements

3. Power Management on the SCC

4. ManyMan - Multi-touch Multicore Manager

5. Current Work: Process Networks on the SCC

6. SVP on the SCC
Chip layout

- Experimental research platform
- $6 \times 4$ Mesh network (24 tiles).
- 2 Cores per tile (48 total).
Chip layout

- Experimental research platform
- $6 \times 4$ Mesh network (24 tiles).
- 2 Cores per tile (48 total).
- 4 Memory controllers.
Chip layout

- Experimental research platform
- $6 \times 4$ Mesh network (24 tiles).
- 2 Cores per tile (48 total).
- 4 Memory controllers.
- A voltage regulation controller (VRC).
  - Dynamic Voltage and Frequency scaling.
Chip layout

- Experimental research platform
- $6 \times 4$ Mesh network (24 tiles).
- 2 Cores per tile (48 total).
- 4 Memory controllers.
- A voltage regulation controller (VRC).
  - Dynamic Voltage and Frequency scaling.
- System Interface (SIF) provides an interface to the outside world.
Chip layout

- Experimental research platform
- $6 \times 4$ Mesh network (24 tiles).
- 2 Cores per tile (48 total).
- 4 Memory controllers.
- A voltage regulation controller (VRC).
  - Dynamic Voltage and Frequency scaling.
- System Interface (SIF) provides an interface to the outside world.
- Control through FPGA and PCIe / Ethernet to MCPC (Linux Box)
The Hardware
Network Layout

- Mesh network ($6 \times 4$)
- 800-1600MHz
- 2Tb/s bisection bandwidth
- XY routing protocol
- 5 mesh cycles per hop (4-stage router)
Tile Layout

- Five port crossbar router.
- Two P54C (Pentium 1) cores.
- L1 instruction cache and L1 data cache (both 16KB) per core.
- 256KB L2 cache per core.
- 16KB shared on-chip memory (*Message Passing Buffer*).
- Configuration register bank.
Core

- P54C (1994)
- 100MHz original, for SCC 100MHz-1GHz
- 3.3V original, for SCC 0.7V-1.14V
- Superscalar (dual integer pipe)
- in-order execution
- Only allows single outstanding memory operation
Memory System

- 4 Memory Controllers, 8 GB per controller (32GB total).
- 32 bit address space for the cores (4GB Max).
Memory System

- 4 Memory Controllers, 8 GB per controller (32GB total).
- 32 bit address space for the cores (4GB Max).
- Lookup tables map core addresses to physical addresses with routing information.
- Resulting in $256 \times 16\text{MB}$ LUT pages.
- LUT mapping can be changed at runtime, by all cores.
Memory: Latency Differences

- Latency and throughput are depending on the distance to the memory controller.
- Average about the same except for upper row of tiles.
Memory: Latency Differences

- Latency and throughput are depending on the distance to the memory controller.
- Average about the same except for upper row of tiles.
- Conclusion: Memory access is non-uniform.
Memory: Page Table Settings

- Different settings:
  - MPBT (Message Passing Buffer Type):
    - Enables Write Combine Buffer (WCB)
    - Bypasses L2 Cache
Memory: Page Table Settings

- **Different settings:**
  - **MPBT (Message Passing Buffer Type):**
    - Enables Write Combine Buffer (WCB)
    - Bypasses L2 Cache
  - **PWT (Page Write Through):**
    - Puts L1 cache in Write Through mode.
Memory: Page Table Settings

- **Different settings:**
  - MPBT (Message Passing Buffer Type):
    - Enables Write Combine Buffer (WCB)
    - Bypasses L2 Cache
  - PWT (Page Write Through):
    - Puts L1 cache in Write Through mode.
  - PCD (Page Cache Disable):
    - Disables L1 and L2 cache.
Memory: Page Table Settings

Different settings:
- MPBT (Message Passing Buffer Type):
  - Enables Write Combine Buffer (WCB)
  - Bypasses L2 Cache
- PWT (Page Write Through):
  - Puts L1 cache in Write Through mode.
- PCD (Page Cache Disable):
  - Disables L1 and L2 cache.
Memory Controller Saturation

- 4 Memory Controllers, 2 banks each, 2 ranks each.
- What if multiple cores access the same memory region?
4 Memory Controllers, 2 banks each, 2 ranks each.

What if multiple cores access the same memory region?

Conclusion: Memory system scales as long as the load is properly divided.

Large memory copy operations can be performed in parallel by multiple Copy Cores.
Memory Controller Saturation

- 4 Memory Controllers, 2 banks each, 2 ranks each.
- What if multiple cores access the same memory region?
- Conclusion: Memory system scales as long as the load is properly divided.
Memory Controller Saturation

- 4 Memory Controllers, 2 banks each, 2 ranks each.
- What if multiple cores access the same memory region?
- Conclusion: Memory system scales as long as the load is properly divided.
- Large memory copy operations can be performed in parallel by multiple Copy Cores.
Message Passing Buffer (MPB)

- 16KB per tile, 384KB total (24 × 16).
- Accessible from all cores
  - Latency 61 cycles (local) plus ≈ 5 cycles per hop in the mesh.

Bypass for local MPB does not work. Latency could have been reduced to ≈ 6 cycles.

Programming:
- Intel’s message passing protocol: RCCE (and iRCCE).
- Direct Access.
Message Passing Buffer (MPB)

- 16KB per tile, 384KB total (24 × 16).
- Accessible from all cores
  - Latency 61 cycles (local) plus ≈ 5 cycles per hop in the mesh.
- Bypass for local MPB does not work
  - Latency could have been reduced to ≈ 6 cycles.
Message Passing Buffer (MPB)

- 16KB per tile, 384KB total (24 × 16).
- Accessible from all cores
  - Latency 61 cycles (local) plus ≈ 5 cycles per hop in the mesh.
- Bypass for local MPB does not work
  - Latency could have been reduced to ≈ 6 cycles.
- Programming:
  - Intel’s message passing protocol: RCCE (and iRCCE).
  - Direct Access.
Message Passing Buffer (MPB)

- 16KB per tile, 384KB total (24 \times 16).
- Accessible from all cores
  - Latency 61 cycles (local) plus \approx 5 cycles per hop in the mesh.
- Bypass for local MPB does not work
  - Latency could have been reduced to \approx 6 cycles.
- Programming:
  - Intel’s message passing protocol: RCCE (and iRCCE).
  - Direct Access.

![Graph](image-url)
Cache Flushing

- No problem for L1:
  - CL1INVMB instruction
  - INVD, WBINVD

Cache flushing is expensive, but required to use cache-able shared memory.

Roy Bakker (UvA - CSA)
Cache Flushing

- No problem for L1:
  - CL1INVMB instruction
  - INVD, WBINVD

- No such instructions for L2:
  - Flush by reading 256KB of data:
    - 1505K cycles *dirty*,
    - 939K cycles *allocated*.

- Optimized:
  - 1275k cycles *dirty*,
  - 574K cycles *allocated*. 
Cache flushing is expensive, but required to use cache-able shared memory.
Outline

1. About me
2. The SCC
3. Power Management on the SCC
   - Power Measurements
4. ManyMan - Multi-touch Multicore Manager
5. Current Work: Process Networks on the SCC
6. SVP on the SCC
Power settings

- Voltage control
  - 6 Voltage Islands (4 tiles, 8 cores each)
  - Voltages from 0.6 to 1.3 Volt (stable from: 0.65625)
  - Adjustable at 0.0625 Volt steps (by hex value, $0.0625 = \frac{0.1}{16}$)
Power settings

- **Voltage control**
  - 6 *Voltage Islands* (4 tiles, 8 cores each)
  - Voltages from 0.6 to 1.3 Volt (stable from: 0.65625)
  - Adjustable at 0.0625 Volt steps (by hex value, $0.0625 = \frac{0.1}{16}$)

- **Frequency control**
  - 24 *Frequency Islands* (1 tile, 2 cores each)
  - Frequency divider (2-16) from global 1600 MHz clock (800, 533, 400, ...)

Roy Bakker (UvA - CSA)
Intel’s preferred way: RCCE

- Requires RCCE
  - RCCE has a rather complicated startup / init
  - Voltage only adjustable by local *Voltage Domain Master*
  - Frequency only adjustable by local *Frequency Domain Master*

Voltages and Frequencies in indexing table:

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>460</td>
</tr>
<tr>
<td>0.8</td>
<td>598</td>
</tr>
<tr>
<td>0.9</td>
<td>644</td>
</tr>
<tr>
<td>1.0</td>
<td>748</td>
</tr>
<tr>
<td>1.1</td>
<td>875</td>
</tr>
<tr>
<td>1.2</td>
<td>1024</td>
</tr>
<tr>
<td>1.3</td>
<td>1198</td>
</tr>
</tbody>
</table>

Remember: Frequencies are 800, 533, 400, 320, 267, 229, 200, ..., 100

Only two voltage settings available 1, 0.

Conclusion: We need to do it better and more easy!
Intel’s preferred way: RCCE

- Requires RCCE
- RCCE has a rather complicated startup / init
- Voltage only adjustable by local Voltage Domain Master
- Frequency only adjustable by local Frequency Domain Master
- Voltages and Frequencies in indexing table:

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>460</td>
</tr>
<tr>
<td>0.8</td>
<td>598</td>
</tr>
<tr>
<td>0.9</td>
<td>644</td>
</tr>
<tr>
<td>1.0</td>
<td>748</td>
</tr>
<tr>
<td>1.1</td>
<td>875</td>
</tr>
<tr>
<td>1.2</td>
<td>1024</td>
</tr>
<tr>
<td>1.3</td>
<td>1198</td>
</tr>
</tbody>
</table>

Remember: Frequencies are 800, 533, 400, 320, 267, 229, 200, ..., 100

Only two voltage settings available: 1.2, 0.8.

Conclusion: We need to do it better and more easy!
Intel’s preferred way: RCCE

- Requires RCCE
  - RCCE has a rather complicated startup / init
  - Voltage only adjustable by local Voltage Domain Master
  - Frequency only adjustable by local Frequency Domain Master
  - Voltages and Frequencies in indexing table:

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>460</td>
</tr>
<tr>
<td>0.8</td>
<td>598</td>
</tr>
<tr>
<td>0.9</td>
<td>644</td>
</tr>
<tr>
<td>1.0</td>
<td>748</td>
</tr>
<tr>
<td>1.1</td>
<td>875</td>
</tr>
<tr>
<td>1.2</td>
<td>1024</td>
</tr>
<tr>
<td>1.3</td>
<td>1198</td>
</tr>
</tbody>
</table>

  - Remember: Frequencies are 800, 533, 400, 320, 267, 229, 200, ..., 100
Intel’s preferred way: RCCE

- Requires RCCE
  - RCCE has a rather complicated startup / init
  - Voltage only adjustable by local *Voltage Domain Master*
  - Frequency only adjustable by local *Frequency Domain Master*
  - Voltages and Frequencies in indexing table:

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>460</td>
</tr>
<tr>
<td>0.8</td>
<td>598</td>
</tr>
<tr>
<td>0.9</td>
<td>644</td>
</tr>
<tr>
<td>1.0</td>
<td>748</td>
</tr>
<tr>
<td>1.1</td>
<td>875</td>
</tr>
<tr>
<td>1.2</td>
<td>1024</td>
</tr>
<tr>
<td>1.3</td>
<td>1198</td>
</tr>
</tbody>
</table>

- Remember: Frequencies are 800, 533, 400, 320, 267, 229, 200, ..., 100
- Only two voltage settings available 1.1, 0.8!
Intel’s preferred way: RCCE

- Requires RCCE
  - RCCE has a rather complicated startup / init
  - Voltage only adjustable by local *Voltage Domain Master*
  - Frequency only adjustable by local *Frequency Domain Master*
  - Voltages and Frequencies in indexing table:

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>460</td>
</tr>
<tr>
<td>0.8</td>
<td>598</td>
</tr>
<tr>
<td>0.9</td>
<td>644</td>
</tr>
<tr>
<td>1.0</td>
<td>748</td>
</tr>
<tr>
<td>1.1</td>
<td>875</td>
</tr>
<tr>
<td>1.2</td>
<td>1024</td>
</tr>
<tr>
<td>1.3</td>
<td>1198</td>
</tr>
</tbody>
</table>

- Remember: Frequencies are 800, 533, 400, 320, 267, 229, 200, ..., 100
- Only two voltage settings available 1.1, 0.8!

- Conclusion: We need to do it better and more easy!
Powerlib

- Working *Powerlib* after a bit of hacking and crashing the SCC (voltage too low for frequency)
- A better table (based on paper by P. Gschwandner):

<table>
<thead>
<tr>
<th>Frequency divider</th>
<th>Frequency</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>800</td>
<td>1.16250</td>
</tr>
<tr>
<td>3</td>
<td>533</td>
<td>0.85625</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>0.75625</td>
</tr>
<tr>
<td>5</td>
<td>320</td>
<td>0.69375</td>
</tr>
<tr>
<td>6</td>
<td>267</td>
<td>0.66875</td>
</tr>
<tr>
<td>7</td>
<td>229</td>
<td>0.65625</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>0.65625</td>
</tr>
<tr>
<td>9-15</td>
<td>...</td>
<td>0.65625</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>0.65625</td>
</tr>
</tbody>
</table>
Powerlib

- Working *Powerlib* after a bit of hacking and crashing the SCC (voltage too low for frequency)
- A better table (based on paper by P. Gschwandner):

<table>
<thead>
<tr>
<th>Frequency divider</th>
<th>Frequency</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>800</td>
<td>1.16250</td>
</tr>
<tr>
<td>3</td>
<td>533</td>
<td>0.85625</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>0.75625</td>
</tr>
<tr>
<td>5</td>
<td>320</td>
<td>0.69375</td>
</tr>
<tr>
<td>6</td>
<td>267</td>
<td>0.66875</td>
</tr>
<tr>
<td>7</td>
<td>229</td>
<td>0.65625</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>0.65625</td>
</tr>
<tr>
<td>9-15</td>
<td>...</td>
<td>0.65625</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>0.65625</td>
</tr>
</tbody>
</table>

- Any core can set V/F values for chip
Powerlib

- Working *Powerlib* after a bit of hacking and crashing the SCC (voltage too low for frequency)
- A better table (based on paper by P. Gschwandner):

<table>
<thead>
<tr>
<th>Frequency divider</th>
<th>Frequency</th>
<th>Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>800</td>
<td>1.16250</td>
</tr>
<tr>
<td>3</td>
<td>533</td>
<td>0.85625</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>0.75625</td>
</tr>
<tr>
<td>5</td>
<td>320</td>
<td>0.69375</td>
</tr>
<tr>
<td>6</td>
<td>267</td>
<td>0.66875</td>
</tr>
<tr>
<td>7</td>
<td>229</td>
<td>0.65625</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>0.65625</td>
</tr>
<tr>
<td>9-15</td>
<td>...</td>
<td>0.65625</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>0.65625</td>
</tr>
</tbody>
</table>

- Any core can set V/F values for chip
- Specify a domain and frequency, the library selects the highest possible frequency.
- Currently only control over Voltage Domains
  - Frequency scaling is not really effective when voltage is too high
  - Should support different frequencies for one Voltage Domain for fine-grained control
Energy Consumption

- Voltage vs. Frequency
- Power vs. Frequency
- FLOPs vs. Frequency
- MFlops/W vs. Frequency
Outline

1. About me
2. The SCC
3. Power Management on the SCC
4. ManyMan - Multi-touch Multicore Manager
5. Current Work: Process Networks on the SCC
6. SVP on the SCC
Requirements

1. Monitor the system:
   - Load, state and power consumption
   - Resource usage for each core
   - Resource usage per task
   - Overview of running, waiting, completed and failed tasks

2. Manage the system:
   - Create a task
   - Start a task on a specified core
   - Migrate tasks
   - Modify frequency and voltage
Requirements

Monitor the system:
- Load, state and power consumption
- Resource usage for each core
- Resource usage per task
- Task output
- Overview of running, waiting, completed and failed tasks
Requirements

1. Monitor the system:
   - Load, state and power consumption
   - Resource usage for each core
   - Resource usage per task
   - Task output
   - Overview of running, waiting, completed and failed tasks

2. Manage the system:
   - Create a task
   - Start a task on a specified core
   - Migrate tasks
   - Modify frequency and voltage
SCC Front-end
ManyMan - Multi-touch Multicore Manager

SCC Back-end

- Written in Python
- SSH for task creation on the SCC from MCPC
  - New `ssh` process for each task
  - SSH connection sharing improves latency
- Monitoring
  - `top`
  - `sccBmc`
- Managing
  - Berkeley Lab Checkpoint/Restart (BLCR)
  - The *Powerlib* is integrated in *ManyMan*
Migration using BLCR

![Graphs showing the comparison of checkpoint and restart times for NFS and /tmp on different data sizes.]

- **Checkpoint NFS** vs. **Restart NFS**
- **Checkpoint /tmp** vs. **Restart /tmp**

Data size (MB) vs. Seconds (log scale)
Frequency Scaling in ManyMan

The Intel SCC

Roy Bakker (UvA - CSA)

The Intel SCC
Frequency Scaling in ManyMan
Outline

1. About me
2. The SCC
3. Power Management on the SCC
4. ManyMan - Multi-touch Multicore Manager
5. Current Work: Process Networks on the SCC
6. SVP on the SCC
Process Networks on the SCC

- Mapping process networks on the SCC cores
- Generated by daedulus/espam toolchain (Leiden)
Process Networks on the SCC

- Mapping process networks on the SCC cores
- Generated by daedalus/espam toolchain (Leiden)
Process Networks on the SCC

- HDPC backend available (to run on SMP hardware using BOOST)
- Need to modify channel implementation and code generation for SCC
Process Networks on the SCC

- HDPC backend available (to run on SMP hardware using BOOST)
- Need to modify channel implementation and code generation for SCC
- Should allow easy migration of light-weight processes (not like ManyMan...)
- Should allow for fine-grained V/F control
Outline

1. About me
2. The SCC
3. Power Management on the SCC
4. ManyMan - Multi-touch Multicore Manager
5. Current Work: Process Networks on the SCC
6. SVP on the SCC
   - Going from an existing implementation
   - Copy Cores
   - Some Results
Reducing Overhead

- The original distributed implementation runs on the SCC without modification.
  - But it could be more efficient.
Reducing Overhead

- The original distributed implementation runs on the SCC without modification.
  - But it could be more efficient.
- The existing implementation causes overhead:
  - Encoding / Decoding with XDR to support heterogeneous platforms.
  - Implicit copy to a communication buffer.
Reducing Overhead

- The original distributed implementation runs on the SCC without modification.
  - But it could be more efficient.
- The existing implementation causes overhead:
  - Encoding / Decoding with XDR to support heterogeneous platforms.
  - Implicit copy to a communication buffer.
- This can be avoided on the SCC:
  - No XDR: SCC cores are homogeneous.
  - Direct communication, data description function.
  - But this is only efficient for (large) arrays.
Implementation

- `mmap` two complete regions of 20 pages (20 × 16MB).
- Source region optimized for reading.
- Destination region optimized for writing (MPBT).
- Sending core requires L2 Flush.
- Receiving core maps MPBT + Write Through, so only CL1INVMB required.
- Beware of WCB issues.
- Bottleneck at cores that need to receive a lot at the same time.
Using Dedicated Copy Cores

- Dedicated copy cores run a *service* to copy memory regions.
- Communication protocol implemented using a ringbuffer in the MPB.
- L2 flush only on sending core, copycore use MPBT + WT.
- Communication is spread among all copy cores.
  - Threshold value for issuing copy cores
  - Split large copy operations over multiple copy cores.
Matrix Multiplication

- Split 2 input matrices to $2 \times 4$ sub matrices.
- Perform 8 matrix multiplications and 4 matrix additions on sub matrices (in parallel).
- Reconstruct new result matrix.
- Split can be performed recursive.

\[
A \times B \rightarrow \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \times \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 \times b_1 & a_1 \times b_2 \\ a_3 \times b_1 & a_3 \times b_2 \end{bmatrix} \begin{bmatrix} a_2 \times b_3 & a_2 \times b_4 \\ a_4 \times b_3 & a_4 \times b_4 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} \rightarrow C
\]
Matrix Multiplication

- Split 2 input matrices to $2 \times 4$ sub matrices.
- Perform 8 matrix multiplications and 4 matrix additions on sub matrices (in parallel).
- Reconstruct new result matrix.
- Split can be performed recursive.

$$A \times B \rightarrow \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \times \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 \times b_1 & a_1 \times b_2 \\ a_3 \times b_1 & a_3 \times b_2 \end{bmatrix} + \begin{bmatrix} a_2 \times b_3 & a_2 \times b_4 \\ a_4 \times b_3 & a_4 \times b_4 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} \rightarrow C$$
Matrix Multiplication - One recursion step

- Baseline: single thread on single core.
Matrix Multiplication - One recursion step

- **Baseline**: single thread on single core.

![Graph showing speedup for different configurations of computing cores for matrix multiplication.](image)
Matrix Multiplication - One recursion step

- Baseline: single thread on single core.
Matrix Multiplication - One recursion step

- Baseline: single thread on single core.

![Graphs showing speedup for different matrix sizes and computing cores.](image)

- 128 × 128
- 256 × 256
- 512 × 512
- 1024 × 1024
Matrix Multiplication - Two recursion steps

- Baseline: single thread on single core.
Matrix Multiplication - Two recursion steps

- Baseline: single thread on single core.

![Graph showing speedup for different core counts and matrix sizes: 1024 x 1024 and 2048 x 2048.](image-url)
Matrix Multiplication - Two recursion steps

- Baseline: single thread on single core.
Matrix Multiplication - Two recursion steps

- Baseline: single thread on single core.

![Graph showing speedup vs number of computing cores for different matrix sizes and core configurations.]
Questions

?

Also feel free to ask off-line
R.Bakker@uva.nl

More info and references:
http://www.science.uva.nl/~bakkerr/