

TDDD56 Programming parallel computers



Parallel Sorting Contest 2014

The winners are ...

Christoph Kessler, IDA,
Linköpings universitet

Parallel Sorting Contest 2014



- Category Konrad Zuse:
-
-
- Category Southfork:
-
-

Congratulations!

... and thanks to all who participated!

C. Kessler, IDA, Linköpings universitet.

2

TDDD56 Programming parallel computers



Wrap-Up and Outlook

Christoph Kessler, IDA,
Linköpings universitet

* Similar as in TDDC78

Lectures (1)



- **Lecture 1:**
Motivation, Multicore architectural concepts and trends.
- **Lecture 2:** Parallel programming with threads and tasks.
- **Lesson 1:** How to measure and visualize performance of parallel programs. CPU lab introduction.
- **Lecture 3:** Shared memory architecture concepts and performance issues*.
- **Lecture 4:** Non-blocking synchronization.
- **Lecture 5-6:** Theory: Design and analysis of parallel algorithms*.
- **Lecture 7:** Parallel sorting algorithms.
- **Lesson 2:** Selected theory exercises.

C. Kessler, IDA, Linköpings universitet.

4

Lectures (2)



...

- **Lecture 8:** Parallelization of sequential programs*
- **Lecture 9:** GPU architecture and trends
- **Lecture 10:** Introduction to CUDA programming
- **Lecture 11:** CUDA programming. GPU lab introduction
- **Lecture 12:** Sorting on GPU. Advanced CUDA issues.
FFT on GPU
- **Lecture 13:** Introduction to OpenCL.
- **Lesson 3:** OpenCL. Shader programming. Exercises.
- **Lecture 14:** High-level Parallel Programming with Skeletons*
- **Lecture 15:** Advanced issues. Wrap-up

C. Kessler, IDA, Linköpings universitet.

5

TDDD56 Programming parallel computers



Outlook

Christoph Kessler, IDA,
Linköpings universitet

Related course

TDDC78 Programming of Parallel Computers - Methods and Tools, 6hp, VT2

- Complement to TDDD56
- Focus on parallel scientific computing and clusters (NSC)
 - Parallel architecture concepts, including interconnection networks, shared and distributed memory architectures
 - Thread programming with OpenMP (Labs)
 - Message passing programming with MPI (Labs)
 - Performance analysis tools (Lab)
 - Parallel algorithms in scientific programming, including basic linear algebra and linear system solving
- Some minor overlap (→ synergy)
 - Shared memory architecture, Theory, Parallelizing loops, Skeleton programming



C. Kessler, IDA, Linköpings universitet

7

TDDD56 Programming parallel computers



Master Thesis Topics Available!

Christoph Kessler, IDA,
Linköpings universitet

Master thesis projects available! (1)

- Multicore programming techniques and tools for...
 - GPU-based systems and GPU clusters
 - Intel SCC 48-core research prototype
 - Movidius Myriad1, Myriad2
 - Planned: Adapteva Epiphany, ARM big.LITTLE, ...
- ...

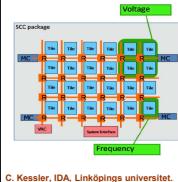


Image source: www.kalray.eu

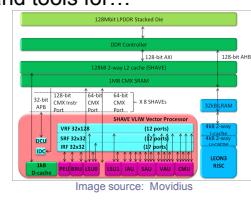
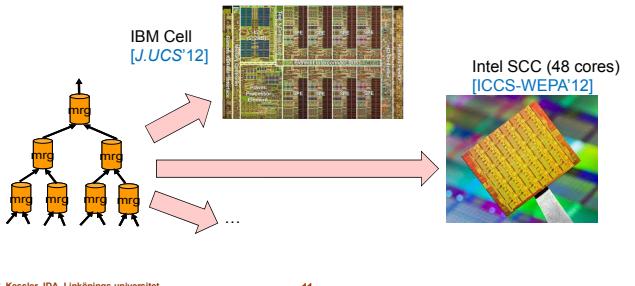


Image source: www.movidius.com



Example: On-Chip Pipelining

- Mapping pipelineable streaming task graphs to manycore systems
 - e.g. pipelined parallel mergesort



C. Kessler, IDA, Linköpings universitet

11

Master thesis projects available! (2)

Parallelization and optimization techniques

- Modeling and optimizing performance and energy efficiency of parallel software
 - ▶ Empirical modeling by adaptive sampling
- Mapping and scheduling tasks to parallel architectures
 - ▶ On-chip pipelining
 - ▶ Crown scheduling



Retargetability / Generic optimization

- Platform modeling
- Micro-benchmarking

C. Kessler, IDA, Linköpings universitet.

10

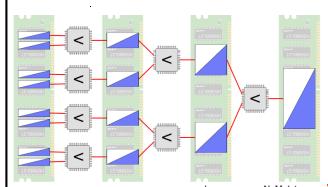
Orchestrating cores and main memory for pipelineable (streaming) computations



Standard model: Dancehall

- Use core-local memory as explicitly managed cache
- Bandwidth to main memory is performance bottleneck
- Easier to implement

Example: Merge tree computation



On-Chip Pipelining [K., Keller HPPC'08]

- All tasks active simultaneously
 - User-level scheduler
 - Round-robin, data driven
- Trades increased on-chip communication for reduced off-chip mem. access
- Example (Cell, PS3):
 - Parallel merging 8M ints on 6 SPEs in 93 ms instead of 225 ms (CellSort)

C. Kessler, IDA, Linköpings universitet.

11

Image source: N. Melot

Generalization

Streaming computations

- Pipelining
- Concurrent execution of all streaming tasks in steady state

Streaming task graphs

- (Acyclic) pipeline graph of streaming tasks
- Producer-consumer communication
- Cf. Kahn Process Networks

C. Kessler, IDA, Linköpings universitet. 13

Generalization

Streaming computations

- Pipelining
- Concurrent execution of all streaming tasks in steady state

Streaming task graphs

- (Acyclic) pipeline graph of streaming tasks
- Producer-consumer communication
- Cf. Kahn Process Networks

Streaming task collections

- Independent streaming tasks
- OR: Steady state of the pipeline (one round): all tasks (instances) considered independent
- Computational load matters, ignore communication for now

C. Kessler, IDA, Linköpings universitet.

Mapping for the steady state

time ↑
One round of the pipeline kernel
User-level round-robin scheduler on each core
P₁ P₂ P₃ ... P_p

C. Kessler, IDA, Linköpings universitet. 15

Now: Moldable (Parallelizable) Tasks

Moldable tasks $j = 1, \dots, n$

- A task j performs fixed **work** $w_{j,0}$
- Can be run with (integer) $w > 1$ cores $t_j(w) = \frac{w_{j,0}}{w e_j(w)}$
- Resource allocation → task **width** w_j
- Limit: **maximum width** W_j : $w_j \leq W_j$ for task j
- Arbitrary scalability functions: **efficiency** $0 < e_j(w) \leq 1$ for $1 \leq w \leq W_j$

Mixed task model

Streaming tasks $j = 1, \dots, n$ can be

- Inherently sequential (max. width $W_j = 1$)
- Malleable with limited scalability (with given $W_j > 1$)
- Malleable with unlimited scalability ($W_j \geq p$)

C. Kessler, IDA, Linköpings universitet.

“Streaming Task Scheduling”

3 static problems to solve (for the steady state of the pipeline):

- **Resource allocation:**
 - Allocate resources (width $w_j \leq W_j$) to each task j
 - Determines its parallel execution time
- **Mapping**
 - Map each task j to w_j specific cores for execution
- **Discrete Frequency Scaling**
 - Select for each task j a frequency f_j in $\{f_1, \dots, f_s\}$ to match a throughput (round makespan) constraint

- Each core then executes its assigned (threads of) tasks at runtime round-robin in a data-driven manner
- Tasks that are not data ready are skipped for the current round

C. Kessler, IDA, Linköpings universitet.

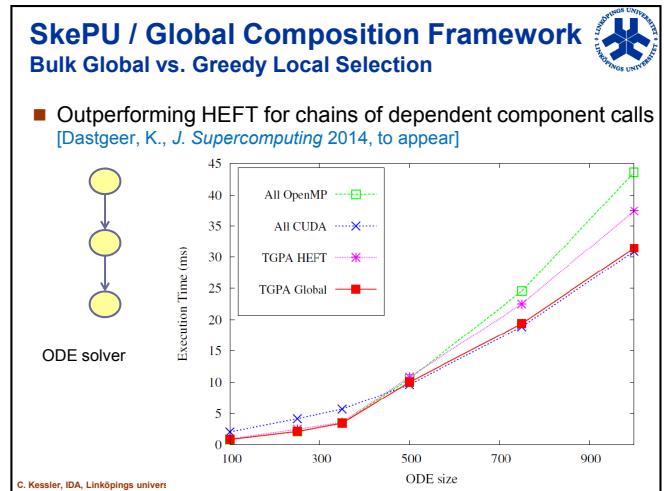
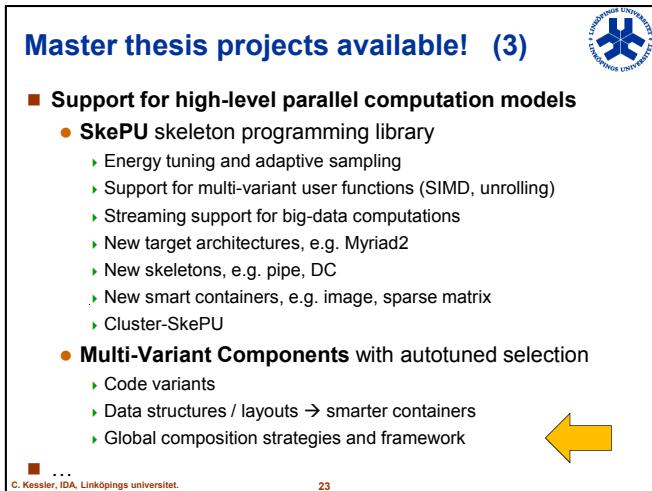
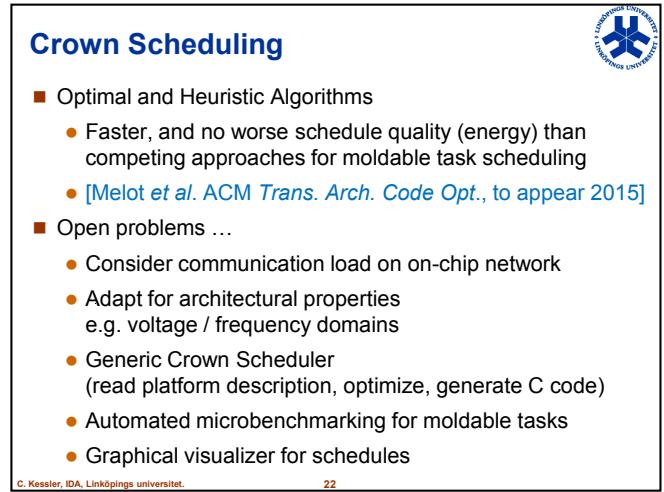
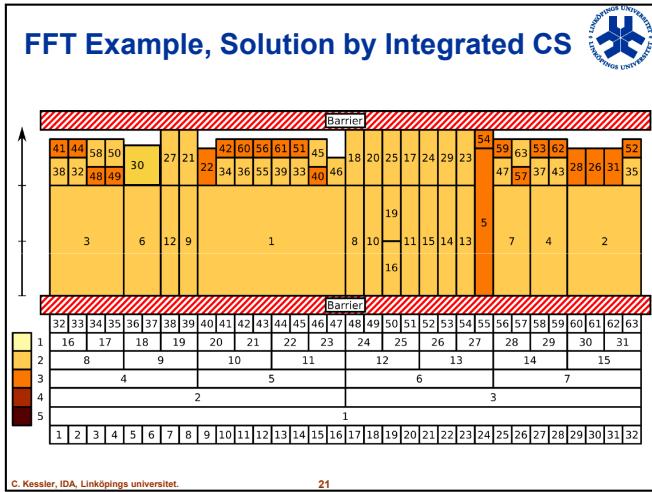
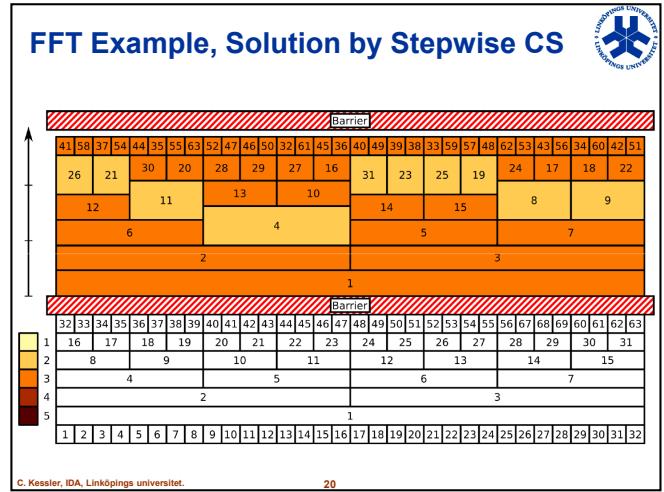
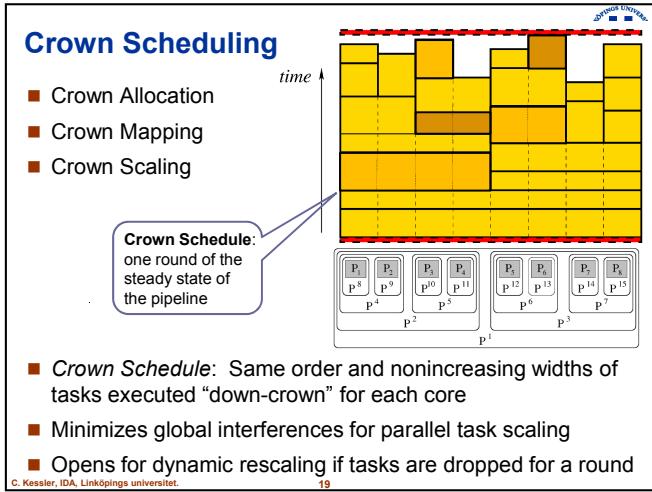
Idea: Constrain Resource Allocation

- Define a hierarchy of processor groups
 - **The Crown**
 - Example: A balanced binary crown over 8 cores

P¹ P² P³ P⁴
P₁ P₂ P₃ P₄
P₁ P₂ P₃ P₄
P₁ P₂ P₃ P₄

- ▶ Core allocations (task widths) w_j must be powers of 2
- ▶ Reduces #possible mapping targets from $2^p - 1$ to $2p - 1$ (= #groups)

C. Kessler, IDA, Linköpings universitet. 18



Master thesis projects available! (4)

- Application porting and algorithm engineering case studies for multicore
 - Medical image processing / Stencil computations
 - Statistics kernels
 - /// IP forwarding data plane on GPUs (algorithms, portability, scheduling)
- Automatic extraction and transformation of parallelism
- ...



Join our research: www.ida.liu.se/~chrke/exjobb

C. Kessler, IDA, Linköpings universitet.

25

The End (?)



*"Now, this is not the end.
It is not even the beginning of the end.
But it is, perhaps, the end of the beginning."*

-- W. Churchill

C. Kessler, IDA, Linköpings universitet.

26



God Jul och Gott Nytt År!