Algorithms and Framework for Energy Efficient Parallel Stream Computing on Many-Core Architectures

Nicolas Melot

Linköping University
Dept. of Computer and Inf. Science
Linköping, Sweden

June 23, 2017

Outline

High performance computing

Constant struggle for performance

High performance computing

Constant struggle for performance: Hollerith census machine

- Census every 10 years.

Picture: "HollerithMachine.CHM" by Adam Schuster

Flickr: Proto IBM. Licensed under CC BY 2.0 via Wikimedia Commons

http://commons.wikimedia.org/wiki/File:HollerithMachine.CHM.jpg#/media/File:HollerithMachine.CHM.jpg

High performance computing

Constant struggle for performance: Hollerith census machine

- Census every 10 years.
- 8 years in 1880.

Picture: “HollerithMachine.CHM” by Adam Schuster

Flickr: Proto IBM. Licensed under CC BY 2.0 via

Wikimedia Commons
http://commons.wikimedia.org/wiki/File:HollerithMachine.CHM.jpg#/media/File:HollerithMachine.CHM.jpg

High performance computing

Constant struggle for performance: Hollerith census machine

- Census every 10 years.
- 8 years in 1880.
- 1 year in 1890.

Picture: "HollerithMachine.CHM" by Adam Schuster

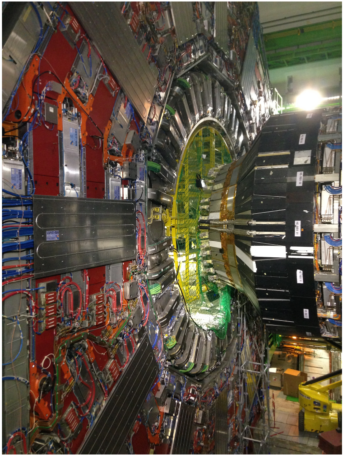
Flickr: Proto IBM. Licensed under CC BY 2.0 via

Wikimedia Commons
http://commons.wikimedia.org/wiki/File:HollerithMachine.CHM.jpg#/media/File:HollerithMachine.CHM.jpg

High performance computing

Constant struggle for performance: Big data.

- Social medias
- Internet of things



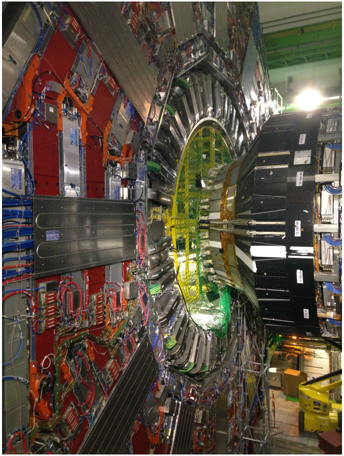
High performance computing

Constant struggle for performance: Big data.

- Social medias
- Internet of things

Applications:

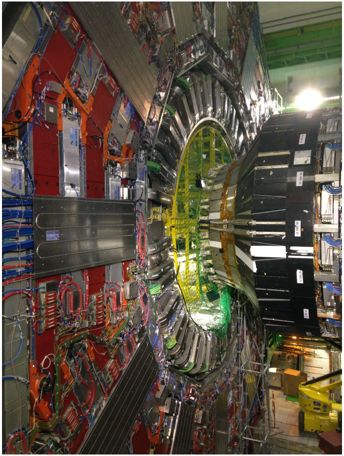
- Scientific computing



Picture: “View inside detector at the CMS cavern LHC CERN” by Tighef

High performance computing

Constant struggle for performance: Big data.



- Social medias
- Internet of things

Applications:

- Scientific computing
- Marketing

Picture: “View inside detector at the CMS cavern LHC CERN” by Tighef

Own work. Licensed under CC BY-SA 3.0 via

Wikimedia Commons

http://commons.wikimedia.org/wiki/File:View_inside_detector_at_the_CMS_cavern_LHC_CERN.jpg#/media/File:

View_inside_detector_at_the_CMS_cavern_LHC_CERN.jpg

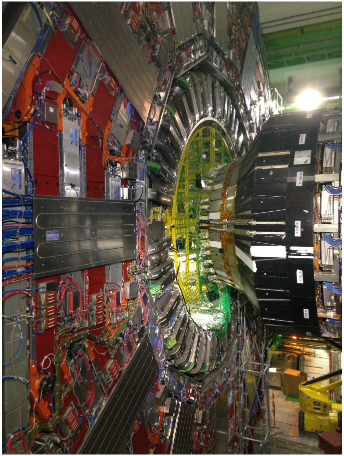
High performance computing

Constant struggle for performance: Big data.

- Social medias
- Internet of things

Applications:

- Scientific computing
- Marketing
- Intelligence



Picture: “View inside detector at the CMS cavern LHC CERN” by Tighef

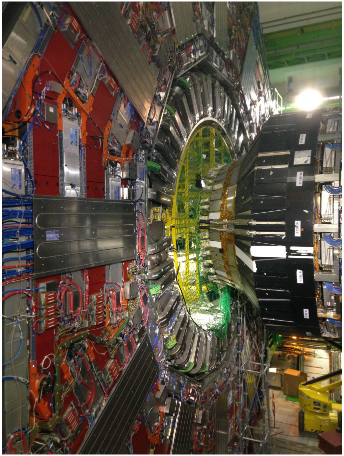
High performance computing

Constant struggle for performance: Big data.

- Social medias
- Internet of things

Applications:

- Scientific computing
- Marketing
- Intelligence (GCHQ)



Picture: "View inside detector at the CMS cavern LHC CERN" by Tighef

Own work. Licensed under CC BY-SA 3.0 via

Wikimedia Commons

http://commons.wikimedia.org/wiki/File:View_inside_detector_at_the_CMS_cavern_LHC_CERN.jpg#/media/File:

View_inside_detector_at_the_CMS_cavern_LHC_CERN.jpg

Accelerate computation

How to improve performance?

Accelerate computation

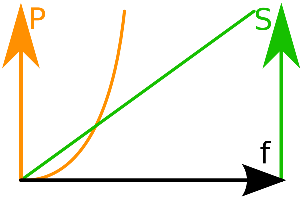
How to improve performance?

- Miniaturize
    - End of Moore's law?

Accelerate computation

How to improve performance?

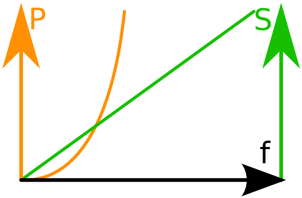
- Miniaturize
  - End of Moore's law?
- Increase frequency
  - Too high energy consumption



Accelerate computation

How to improve performance?

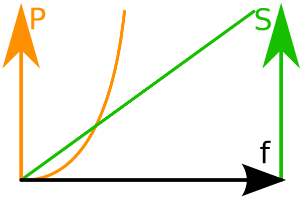
- Miniaturize
  - End of Moore's law?
- Increase frequency
  - Too high energy consumption
- Parallel programming



Accelerate computation

How to improve performance?

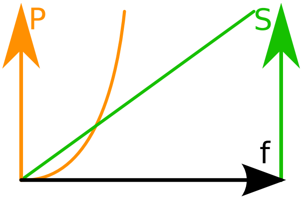
- Miniaturize
  - End of Moore's law?
- Increase frequency
  - Too high energy consumption
- Parallel programming
  - Better energy consumption



Accelerate computation

How to improve performance?

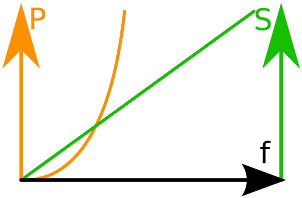
- Miniaturize
  - End of Moore's law?
- Increase frequency
  - Too high energy consumption
- Parallel programming
  - Better energy consumption
  - Very challenging
    - Instruction-Level parallelism Wall



Accelerate computation

How to improve performance?

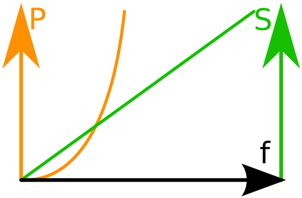
- Miniaturize
  - End of Moore's law?
- Increase frequency
  - Too high energy consumption
- Parallel programming
  - Better energy consumption
  - Very challenging
    - Instruction-Level parallelism Wall
    - Scalability issues for consistent shared memory



Accelerate computation

How to improve performance?

- Miniaturize
  - End of Moore's law?
- Increase frequency
  - Too high energy consumption
- Parallel programming
  - Better energy consumption
  - Very challenging
    - Instruction-Level parallelism Wall
    - Scalability issues for consistent shared memory
    - Von-Neumann bottleneck



Streaming computation

Streaming

- Software pipelining
- Tasks execute in parallel in steady-state

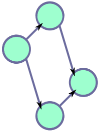


Figure: Streaming taskgraph.

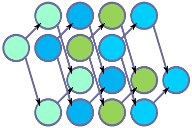


Figure: Pipelined execution of streaming taskgraph.

Streaming computation

Streaming

- Software pipelining
- Tasks execute in parallel in steady-state

Static scheduling

- Moldable tasks
- Steady state
- Throughput constraint
- Optimize energy

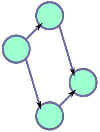


Figure: Streaming taskgraph.

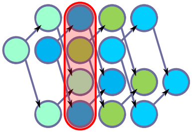


Figure: Steady state of the streaming pipeline.

Streaming computation

Streaming

- Software pipelining
- Tasks execute in parallel in steady-state

Static scheduling

- Moldable tasks
- Steady state
- Throughput constraint
- Optimize energy

Streaming Task Collection

- Independent tasks
- Balance workload
- No communication cost

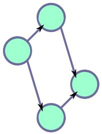


Figure: Streaming taskgraph.

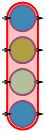


Figure: Independent tasks in the steady state.

Platform model

Platform

- *p* uniform processors
- Discrete frequency set *F*
  - Applied to individual cores
  - Voltage by auto-co-scaling
  - Can change dynamically any time

Power model

- Dynamic power function of frequency
  - Analytic function or measurements
  - No restriction
  - Replaceable
- Energy linear in time, power and number of processors running

Task model

Moldable task j

- Fixed work τ_j
- Allocation: run on $w_j \geq 1$ cores
  - Maximum W_j: $w_j \leq W_j$
- Arbitrary efficiency function
  - $0 < e_j(q) \leq 1$ for $1 \leq q \leq W_j$
  - No convexity, monotony or continuity constraint
- Time proportional to work, parallel degree and frequency

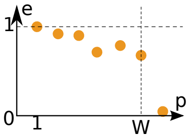


Figure: Arbitrary efficiency function.

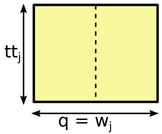
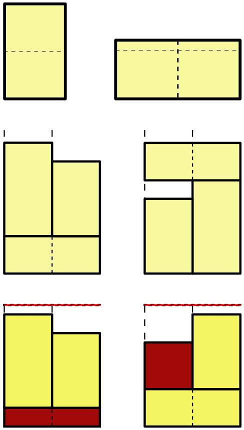


Figure: Moldable task.

Problem formulation

3 static problems

- Resource allocation
  - Find $w_j \leq min(p, W_j)$ for each task j
  - Define execution time of tasks j
- Task mapping to cores
  - Assign tasks to a subset of cores $1..p$
- Discrete frequency scaling
  - Assign tasks a frequency level in F
  - Respect a makespan constraint M
- Repeated execution of a task sequence
- Non data-ready tasks are delayed to the next round

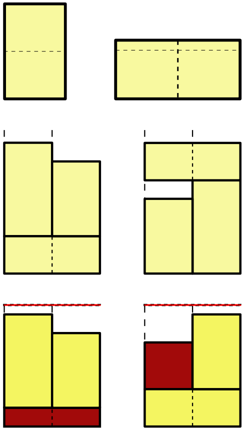


Problem formulation

3 static problems

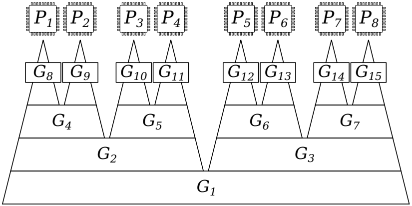
- Resource allocation
  - Find $w_j \leq min(p, W_j)$ for each task j
  - Define execution time of tasks j
- Task mapping to cores
  - Assign tasks to a subset of cores $1..p$
- Discrete frequency scaling
  - Assign tasks a frequency level in F
  - Respect a makespan constraint M
- Repeated execution of a task sequence
- Non data-ready tasks are delayed to the next round

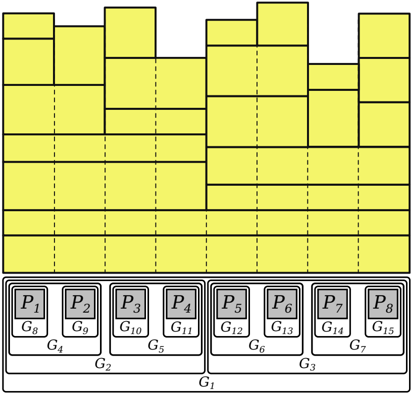
All steps influence each other



Crown scheduling

Restrict allocation and mapping to $O(p)$ processor subsets (groups)

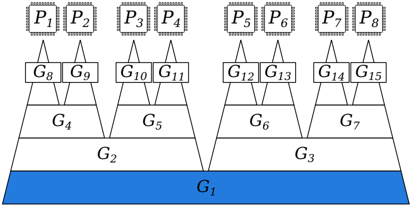


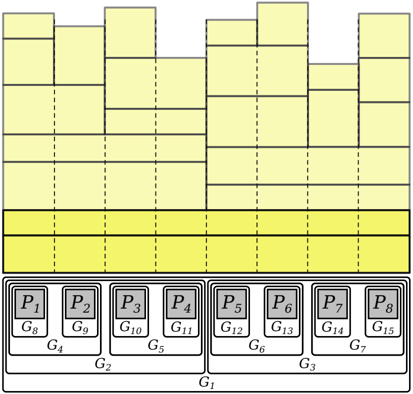


- Tasks must be allocated as many cores as the size of a group
- Reduce possible mapping targets from $2^p - 1$ groups to $2p - 1$

Crown scheduling

Restrict allocation and mapping to $O(p)$ processor subsets (groups)

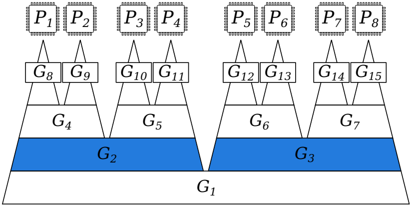


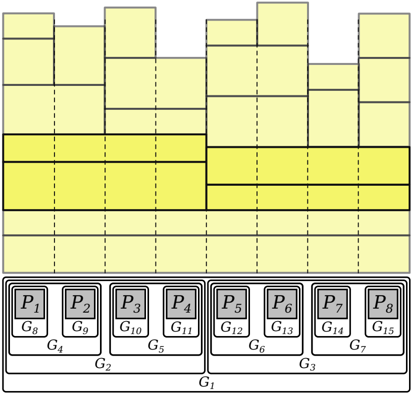


- Tasks must be allocated as many cores as the size of a group
- Reduce possible mapping targets from $2^p - 1$ groups to $2p - 1$

Crown scheduling

Restrict allocation and mapping to $O(p)$ processor subsets (groups)

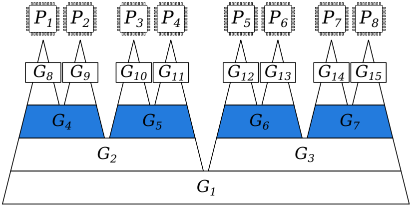


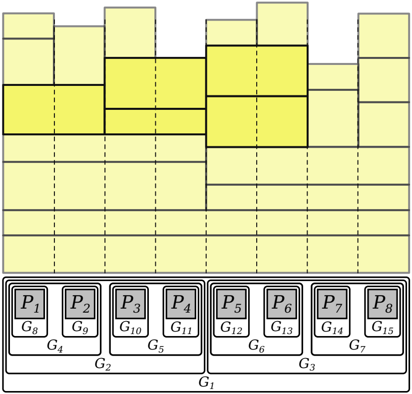


- Tasks must be allocated as many cores as the size of a group
- Reduce possible mapping targets from $2^p - 1$ groups to $2p - 1$

Crown scheduling

Restrict allocation and mapping to $O(p)$ processor subsets (groups)

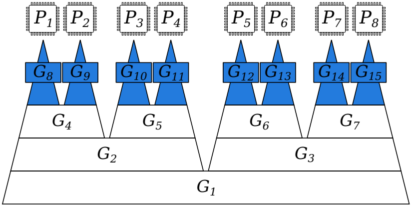


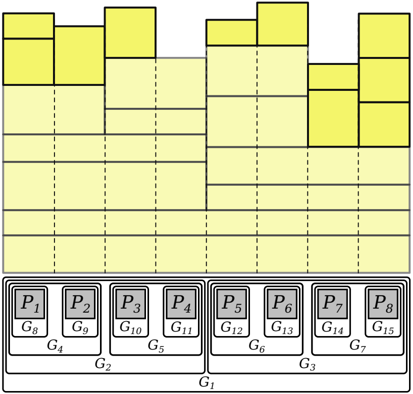


- Tasks must be allocated as many cores as the size of a group
- Reduce possible mapping targets from $2^p - 1$ groups to $2p - 1$

Crown scheduling

Restrict allocation and mapping to $O(p)$ processor subsets (groups)



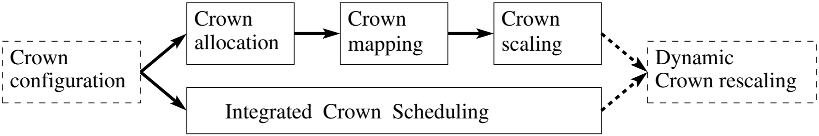


- Tasks must be allocated as many cores as the size of a group
- Reduce possible mapping targets from $2^p - 1$ groups to $2p - 1$

Crown scheduling

Computing a crown schedule

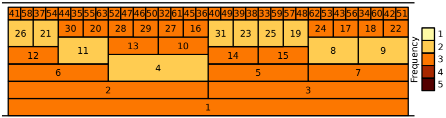
- Separated or integrated phases

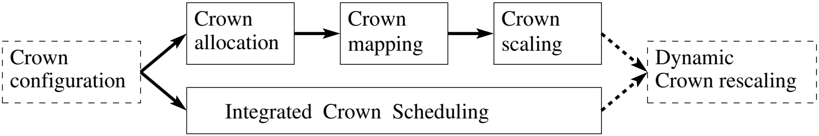


- ILP formulations for each step and for integrated scheduler by (Kessler et al. [2013])
  - Phase separation prevents compromises
  - Phase integrated constrained by the crown structure
  - Slow and limited in input problem size

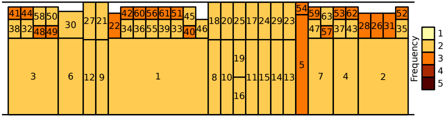
Crown scheduling

Phase-separated





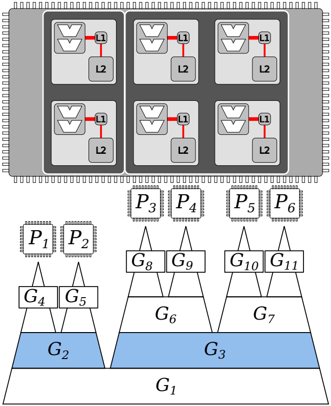
Integrated



Crown Extensions

Adapt to realistic processors

- $p \neq 2^i$
- Constraints: frequency islands

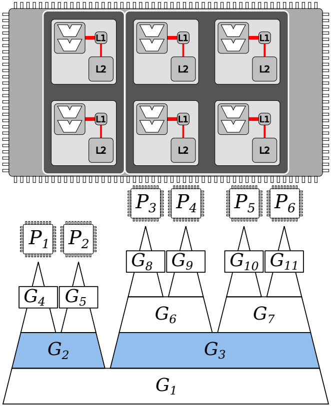


Crown Extensions

Adapt to realistic processors

- $p \neq 2^i$
- Constraints: frequency islands

Crown Configuration



Crown Extensions

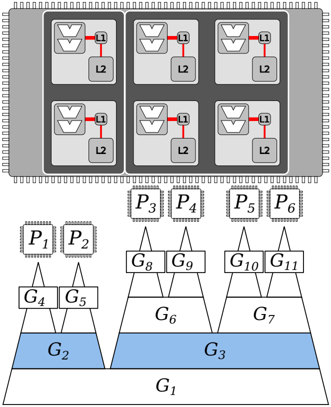
Adapt to realistic processors

- $p \neq 2^i$
- Constraints: frequency islands

Crown Configuration

Crown Consolidation

- Account for idle energy
- Switch unused cores off



Crown Extensions

Adapt to realistic processors

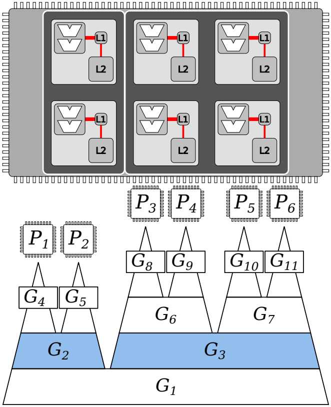
- $p \neq 2^i$
- Constraints: frequency islands

Crown Configuration

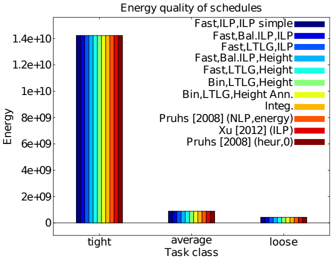
Crown Consolidation

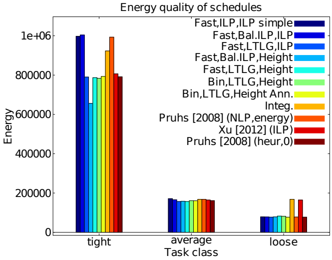
- Account for idle energy
- Switch unused cores off

Provable approximation

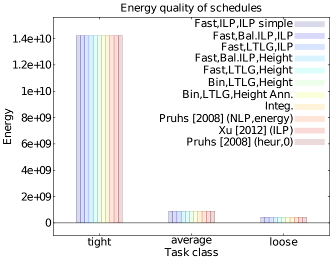


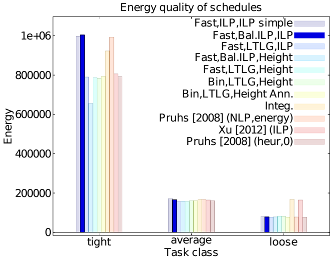
Voltage Islands topology influence

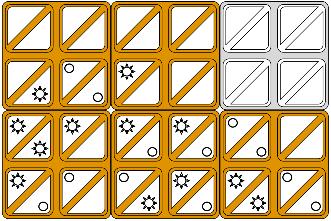




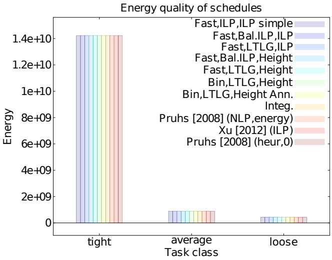
Voltage Islands topology influence

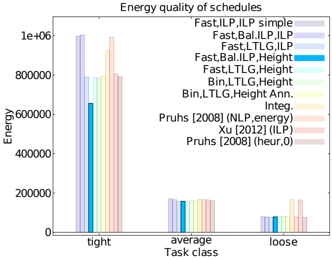


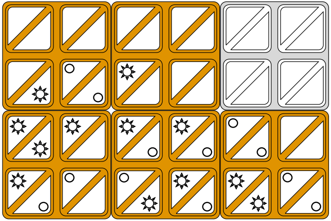


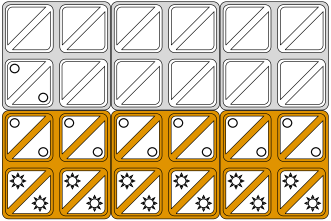


Voltage Islands topology influence









Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

- *External* algorithm

Example: Mergesort

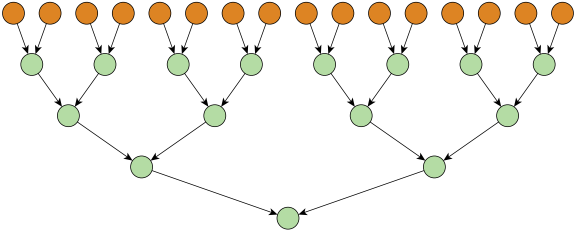
Developed by JohN Von Neumann in 1945 (Knuth [1998])

- *External* algorithm
- Limits use of slow memories

Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

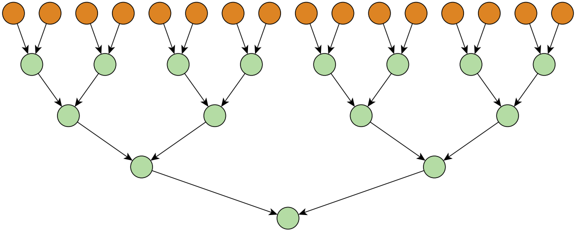
- *External* algorithm
- Limits use of slow memories
- Stream program: tree structure



Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

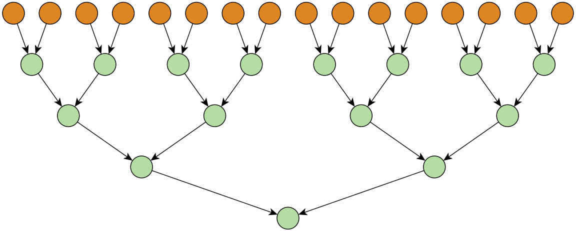
- *External* algorithm
- Limits use of slow memories
- Stream program: tree structure
  - Leaves: presort (non-streamed)



Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

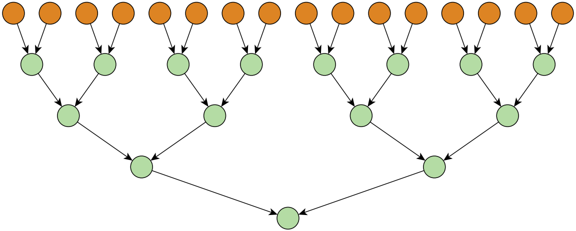
- *External* algorithm
- Limits use of slow memories
- Stream program: tree structure
  - Leaves: presort (non-streamed)
  - other: merge (streamed)



Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

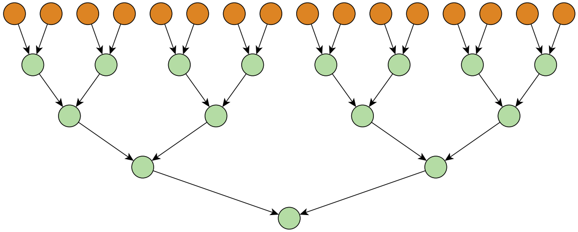
- *External* algorithm
- Limits use of slow memories
- Stream program: tree structure
  - Leaves: presort (non-streamed)
  - other: merge (streamed)
  - Root task: biggest workload



Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

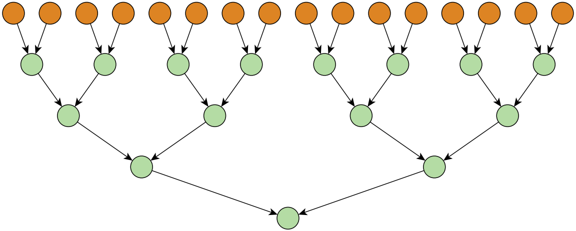
- *External* algorithm
- Limits use of slow memories
- Stream program: tree structure
  - Leaves: presort (non-streamed)
  - other: merge (streamed)
  - Root task: biggest workload
  - 2^{nd} level tasks: half workload of root task



Example: Mergesort

Developed by JohN Von Neumann in 1945 (Knuth [1998])

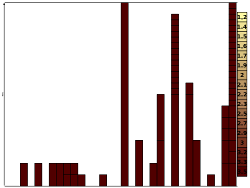
- *External* algorithm
- Limits use of slow memories
- Stream program: tree structure
  - Leaves: presort (non-streamed)
  - other: merge (streamed)
  - Root task: biggest workload
  - 2^{nd} level tasks: half workload of root task



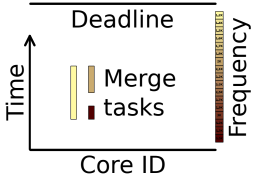
https://www.youtube.com/watch?v=XaqR3G_NVoo

Island-aware scheduling: mergesort

All cores in one island: 100%

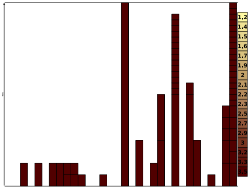


Legend

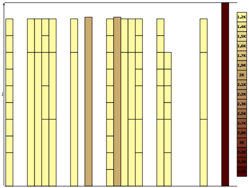


Island-aware scheduling: mergesort

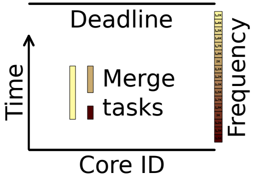
All cores in one island: 100%



Individual cores: 30%

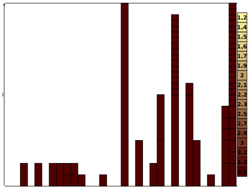


Legend

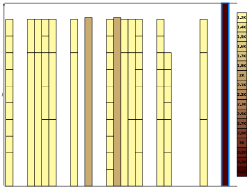


Island-aware scheduling: mergesort

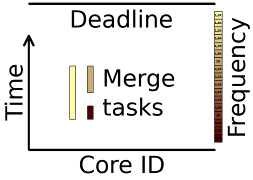
All cores in one island: 100%



Individual cores: 30%

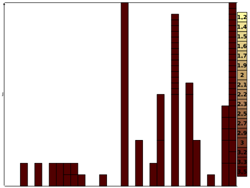


Legend

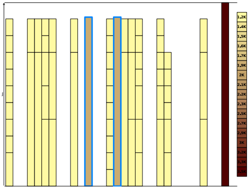


Island-aware scheduling: mergesort

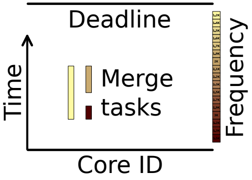
All cores in one island: 100%



Individual cores: 30%

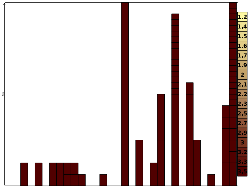


Legend

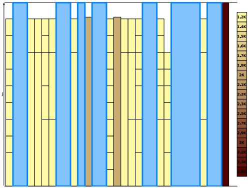


Island-aware scheduling: mergesort

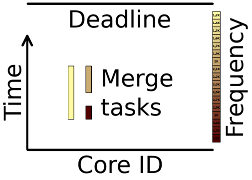
All cores in one island: 100%



Individual cores: 30%

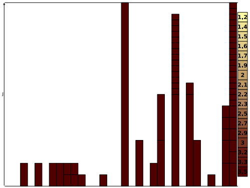


Legend

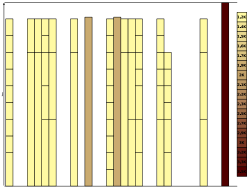


Island-aware scheduling: mergesort

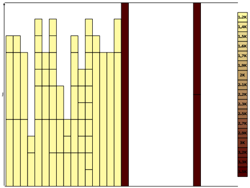
All cores in one island: 100%



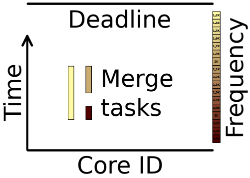
Individual cores: 30%



2 islands of 16 cores: 47%

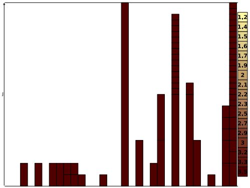


Legend

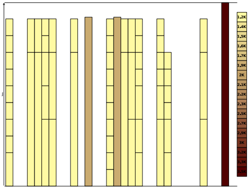


Island-aware scheduling: mergesort

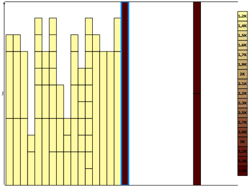
All cores in one island: 100%



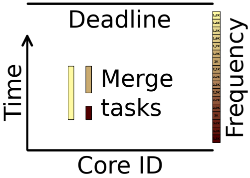
Individual cores: 30%



2 islands of 16 cores: 47%

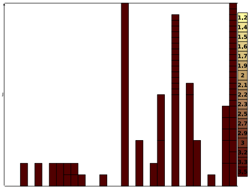


Legend

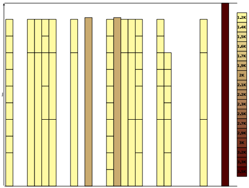


Island-aware scheduling: mergesort

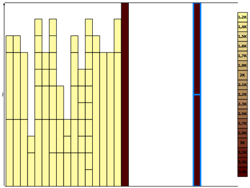
All cores in one island: 100%



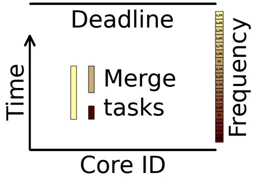
Individual cores: 30%



2 islands of 16 cores: 47%

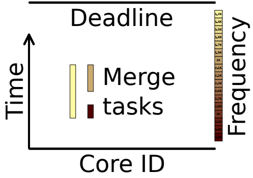


Legend

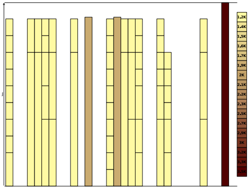


Island-aware scheduling: mergesort

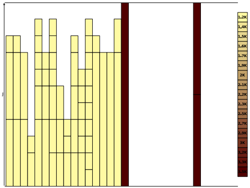
Legend



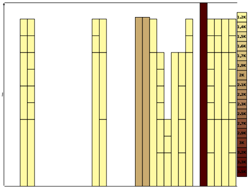
Individual cores: 30%



2 islands of 16 cores: 47%

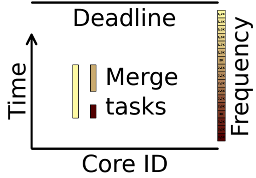


Islands of 2 cores: 33%

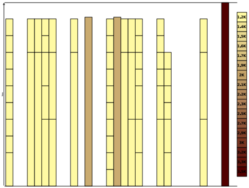


Island-aware scheduling: mergesort

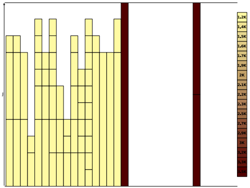
Legend



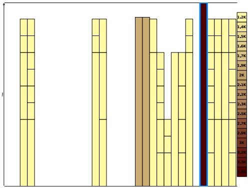
Individual cores: 30%



2 islands of 16 cores: 47%

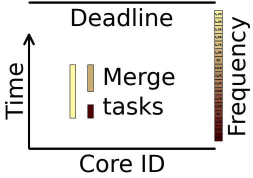


Islands of 2 cores: 33%

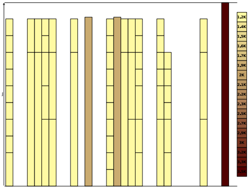


Island-aware scheduling: mergesort

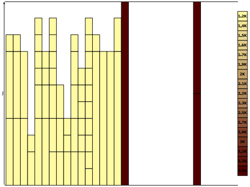
Legend



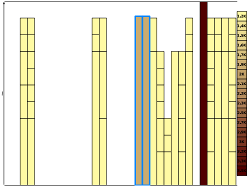
Individual cores: 30%



2 islands of 16 cores: 47%

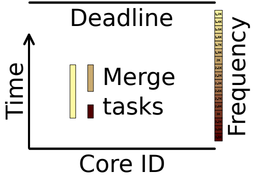


Islands of 2 cores: 33%

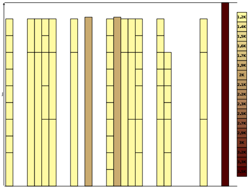


Island-aware scheduling: mergesort

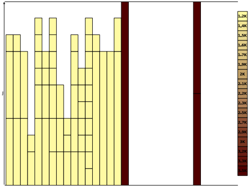
Legend



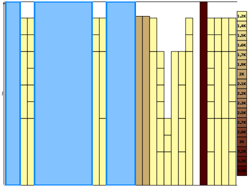
Individual cores: 30%

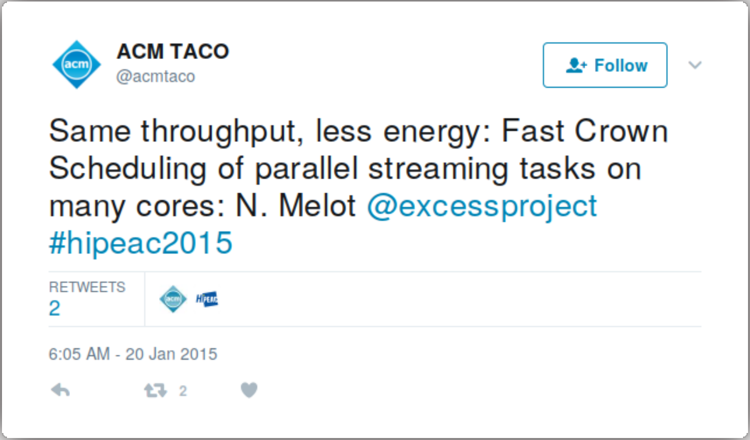


2 islands of 16 cores: 47%



Islands of 2 cores: 33%




ACM TACO
@acmtaco
Follow
Same throughput, less energy: Fast Crown Scheduling of parallel streaming tasks on many cores: N. Melot @excessproject #hipeac2015
RETWEETS
2
6:05 AM - 20 Jan 2015
2

Drake

Stream programming framework

- On-chip pipelining

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling

Drake

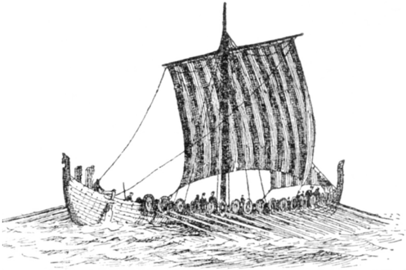
Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling
- Scheduling experiments (Melot et al. [2015])

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling
- Scheduling experiments (Melot et al. [2015])



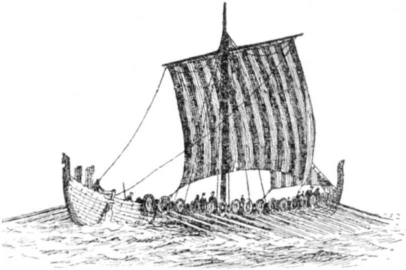
Drake: derived from Schedeval (Janzén [2014])

Separate roles in an application

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling
- Scheduling experiments (Melot et al. [2015])



Drake: derived from Schedeval (Janzén [2014])

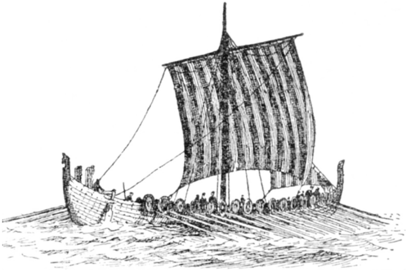
Separate roles in an application

- Stream topology

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling
- Scheduling experiments (Melot et al. [2015])



Drake: derived from Schedeval (Janzén [2014])

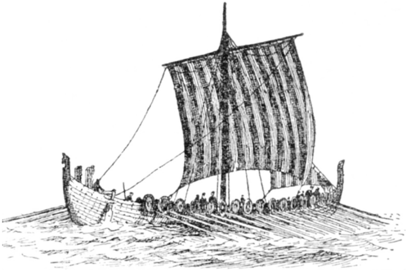
Separate roles in an application

- Stream topology
- Tasks' source code

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling
- Scheduling experiments (Melot et al. [2015])



Drake: derived from Schedeval (Janzén [2014])

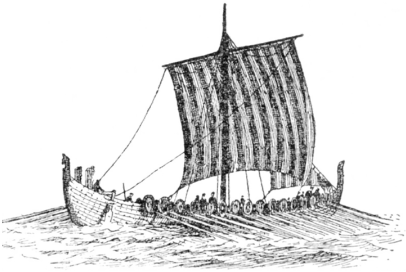
Separate roles in an application

- Stream topology
- Tasks’ source code
- Target platform-specifics

Drake

Stream programming framework

- On-chip pipelining
- Moldable tasks
- Frequency scaling
- Scheduling experiments (Melot et al. [2015])



Drake: derived from Schedeval (Janzén [2014])

Separate roles in an application

- Stream topology
- Tasks' source code
- Target platform-specifics
- Host application

Drake C Streaming Framework

Takes code to execute on target platform

- Application-specific: Mergesort, FFT, etc.

Drake C Streaming Framework

Takes code to execute on target platform

- Application-specific: Mergesort, FFT, etc.
- Platform-specific: SCC, Xeon, MPI, etc.
  - Message passing

Drake C Streaming Framework

Takes code to execute on target platform

- Application-specific: Mergesort, FFT, etc.
- Platform-specific: SCC, Xeon, MPI, etc.
  - Message passing
  - Frequency scaling

Drake C Streaming Framework

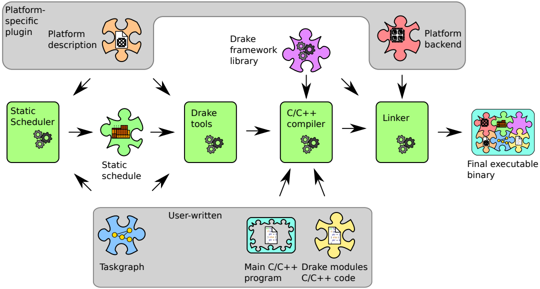
Takes code to execute on target platform

- Application-specific: Mergesort, FFT, etc.
- Platform-specific: SCC, Xeon, MPI, etc.
  - Message passing
  - Frequency scaling
- Generate executable with monitoring.

Drake C Streaming Framework

Takes code to execute on target platform

- Application-specific: Mergesort, FFT, etc.
- Platform-specific: SCC, Xeon, MPI, etc.
  - Message passing
  - Frequency scaling
- Generate executable with monitoring.



Conclusion

Challenges in high-performance parallel computing

- Programming difficulties

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Stream programming

- Mature research

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Stream programming

- Mature research
- No need of coherent shared memory

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Stream programming

- Mature research
- No need of coherent shared memory
- Help reduce von Neumann bottleneck

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Stream programming

- Mature research
- No need of coherent shared memory
- Help reduce von Neumann bottleneck

Our contributions

- On-chip pipelining

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Stream programming

- Mature research
- No need of coherent shared memory
- Help reduce von Neumann bottleneck

Our contributions

- On-chip pipelining
- Crown Scheduling for Moldable Streaming tasks

Conclusion

Challenges in high-performance parallel computing

- Programming difficulties
- Lack of scalability of architectures
- Von Neumann bottleneck

Stream programming

- Mature research
- No need of coherent shared memory
- Help reduce von Neumann bottleneck

Our contributions

- On-chip pipelining
- Crown Scheduling for Moldable Streaming tasks
- Drake Streaming Framework

Future work

Investigate more scheduling techniques

- Minimize communication costs

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures
- Implement Synchronous Data Flow for Drake

Future work

Investigate more scheduling techniques

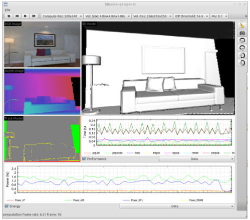
- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures
- Implement Synchronous Data Flow for Drake
- Optimize on-chip memory usage

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures
- Implement Synchronous Data Flow for Drake
- Optimize on-chip memory usage

Simultaneous Localization And Mapping

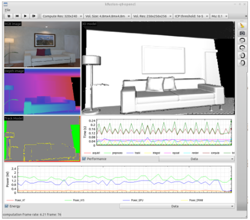


Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures
- Implement Synchronous Data Flow for Drake
- Optimize on-chip memory usage

Simultaneous Localization And Mapping



Other applications:

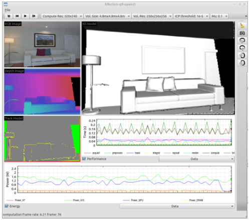
- Machine learning

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures
- Implement Synchronous Data Flow for Drake
- Optimize on-chip memory usage

Simultaneous Localization And Mapping



Other applications:

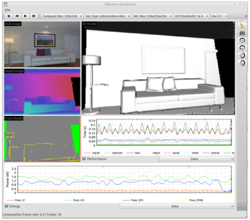
- Machine learning
- Provable High Performance Computing

Future work

Investigate more scheduling techniques

- Minimize communication costs
- Heterogeneous platforms
- Quasi-static scheduling
- Schedule main memory accesses
- Port Drake to mores architectures
- Implement Synchronous Data Flow for Drake
- Optimize on-chip memory usage

Simultaneous Localization And Mapping



Other applications:

- Machine learning
- Provable High Performance Computing
- Aeronautic

Questions

Thank you for your attention.

Bibliography

Johan Janzén. Evaluation of energy-optimizing scheduling algorithms for streaming computations on massively parallel multicore architectures. Master's thesis, Linköping University, 2014. URL http://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A756758.

Christoph Kessler, Nicolas Melot, Patrick Eitschberger, and Jörg Keller. Crown Scheduling: Energy-Efficient Resource Allocation, Mapping and Discrete Frequency Scaling for Collections of Malleable Streaming Tasks. In *Proc. of 23rd Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS 2013)*, 2013.

Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998. ISBN 0-201-89685-0.

Nicolas Melot, Johan Janzen, and Christoph Kessler. Mimer and Schedeval: Tools for Comparing Static Schedulers for Streaming Applications on Manycore Architectures. In *Parallel Processing Workshops (ICPPW), IEEE*, pages 146–155, Sept 2015. doi: 10.1109/ICPPW.2015.24.