Playing Alarik

Architecture and Performance

A first assessment by

Joachim Hein
SNIC Parallel Expert
Lunarc, Lund University

Outline

- Hardware overview
  - Hardware overview
  - AMD's Interlagos CPUs
- Performance
  - Memory bandwidth
  - Interconnect
  - Linpack (BLAS)
- Applications
  - OpenFOAM
  - GROMACS

HARDWARE

May I introduce: Alarik

Current hardware

- 208 compute nodes
- 3328 processing cores in total

- Dual processor nodes: AMD Opteron 6220 "*Interlagos*"
  - 8 compute cores per processor, 16 per node
  - 3 GHz clock frequency, highest AMD offers
  - Up from 8 cores/note on Platon
- DDR3 1600 MHz main memory
  - 70 nodes with 64GB main memory, 4GB per core
  - 138 nodes with 32GB main memory, 2GB per core
  - up from 24 GB/node on Platon

Forthcoming hardware: Large memory and high core-count nodes

- 4 Nodes to be installed in spring 2012

- Four AMD Opteron processors "*Interlagos*"
  - 12 compute cores per processor, **48 cores per node**
  - 2.6 GHz clock frequency
- **128 GB of main memory** per node

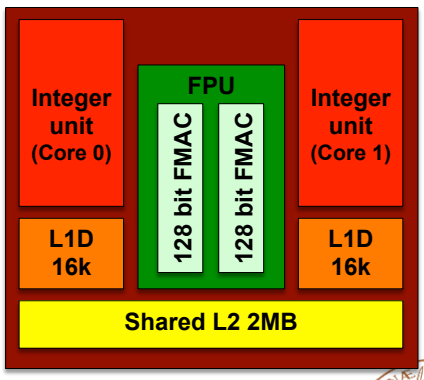
- Ideal for:
  - **threaded applications** (e.g. Posix, OpenMP, ...)
  - Applications requiring **large amounts of main memory**

Infiniband Interconnect

- Mellanox 4x QDR Infiniband switch network
- Single connection per node
- Fat tree with 2 levels of hierarchy
- Three head switches to improve bi-sectional bandwidth

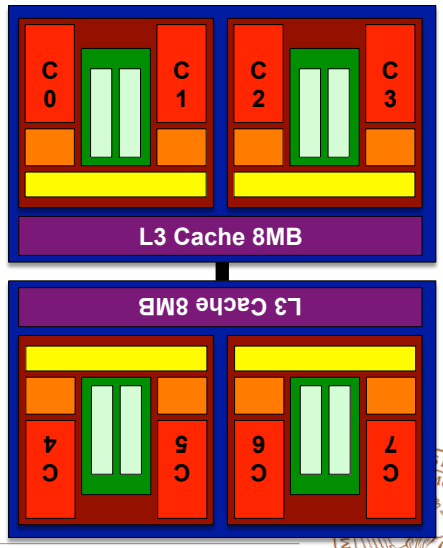
AMD “Bulldozer” module (schematic)

- 2 Integer Cores
- private data L1 each
- 1 shared FPU
  - 2 FPU pipes
  - 128 bit registers
  - each FPU: FMA4/cycle
  - SSE and AVX inst.
- Cores share L2



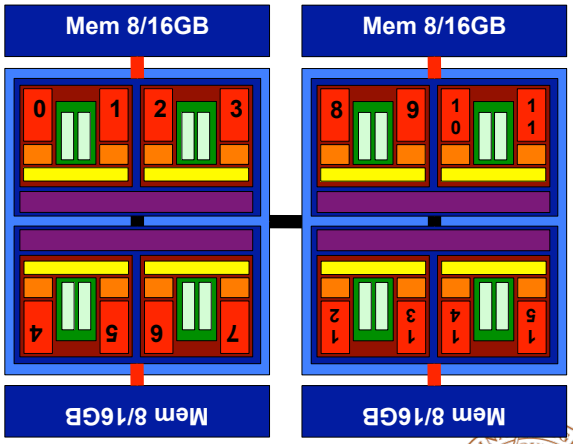
AMD “Interlagos” processor Opteron 6220

- 4 Bulldozer modules
- In 2 groups (dyes)
  - Shared L3 per group
  - Connected via internal hyper link
- Total:
  - 8 Integer cores
  - arranged in 4 Modules
  - sharing 4 FPUs
  - 3 levels of cache



Structure of an Alarik node

- 2 Processors
- 4 dual module groups (dyes)
- Memory attached to these groups (CC-NUMA architecture)
- Hypertransport
- 16 cores
- 8 FPUs
- A lot of structure that affects performance!



MOVING DATA

Investigating Memory bandwidth

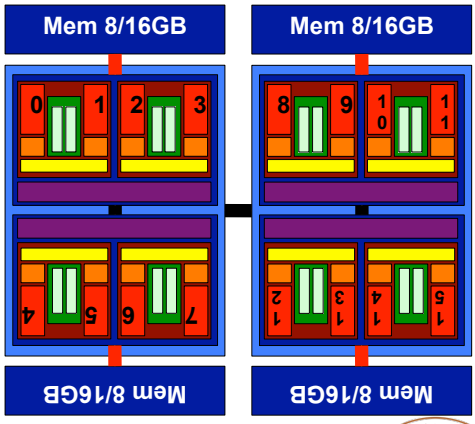
Using local memory:

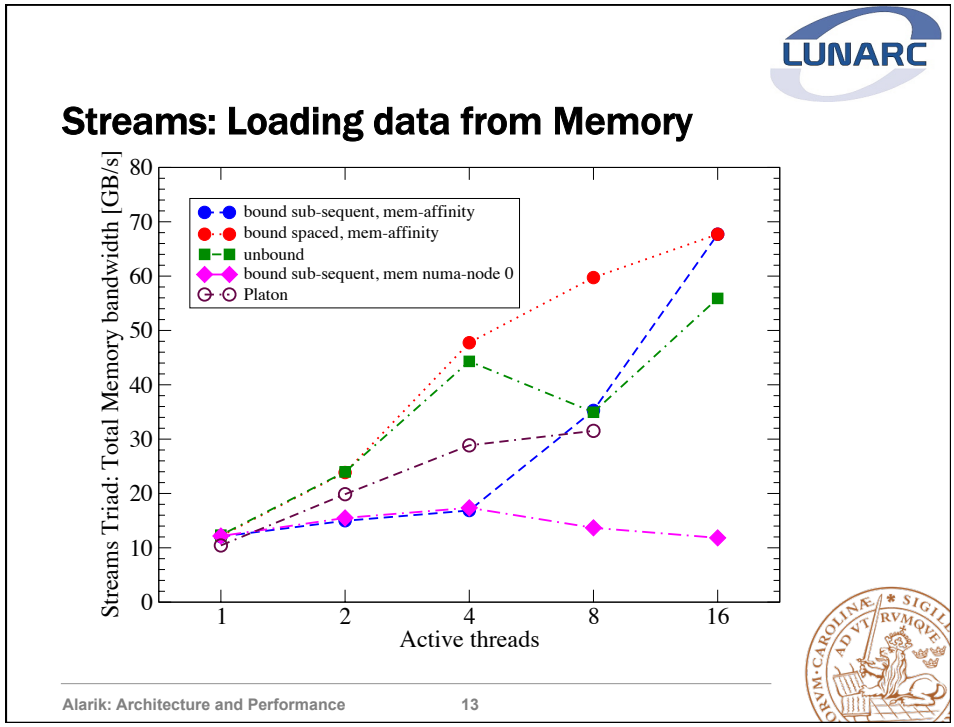
- Processor ordering:
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, …
  - 0, 8, 4, 12, 1, 5, 9, 13, …
  - Let OS choose & migrate

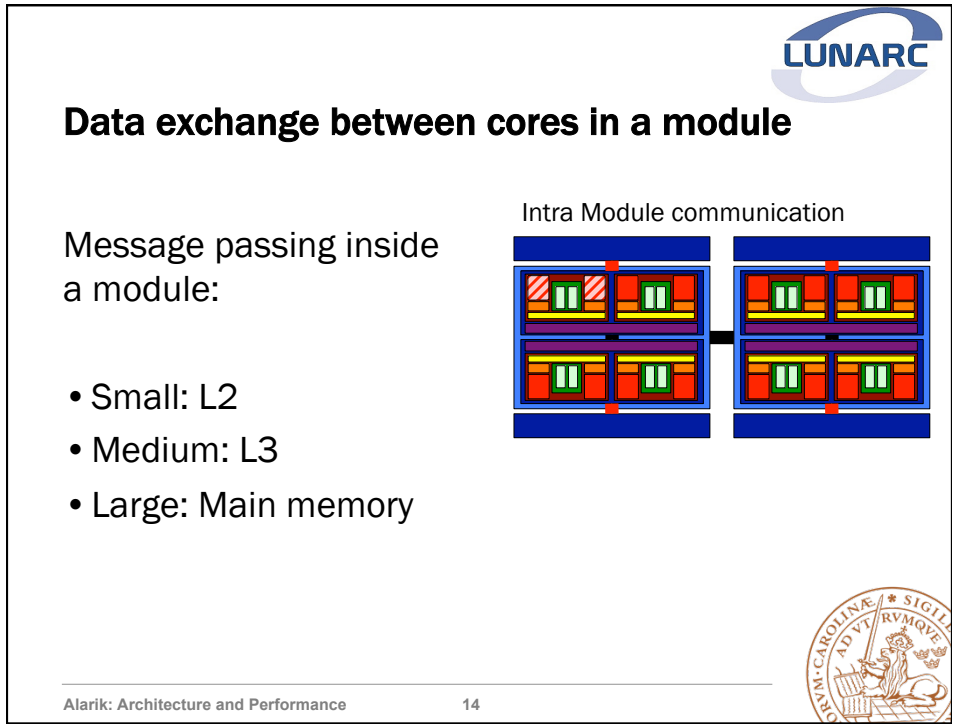
First memory bus:

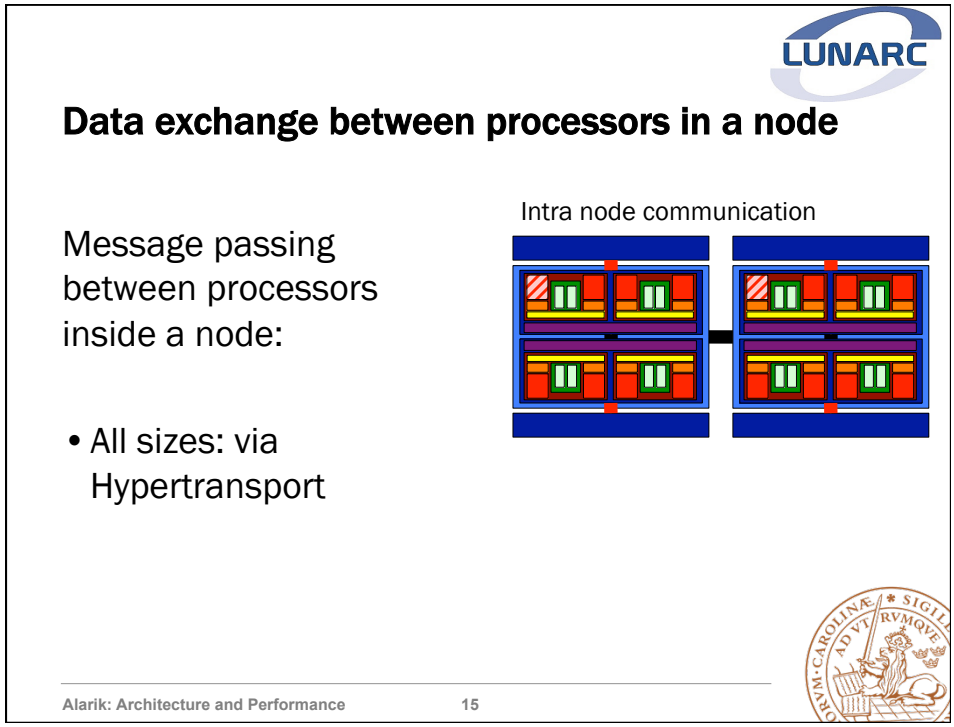
- Processor ordering:
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, …

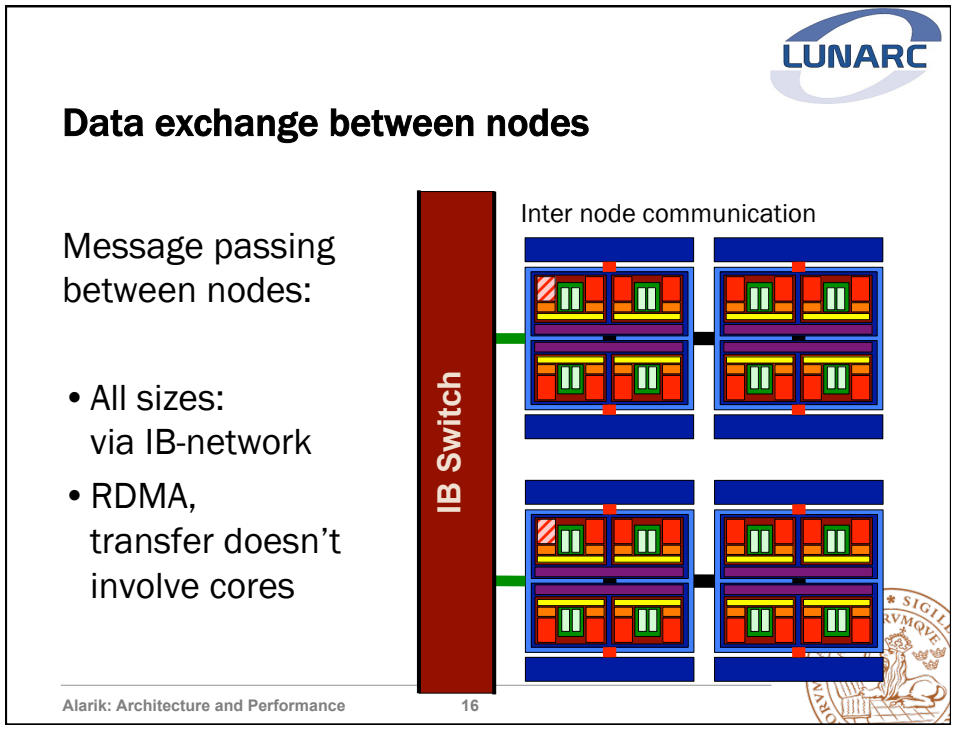
- Open64 compiler, OpenMP



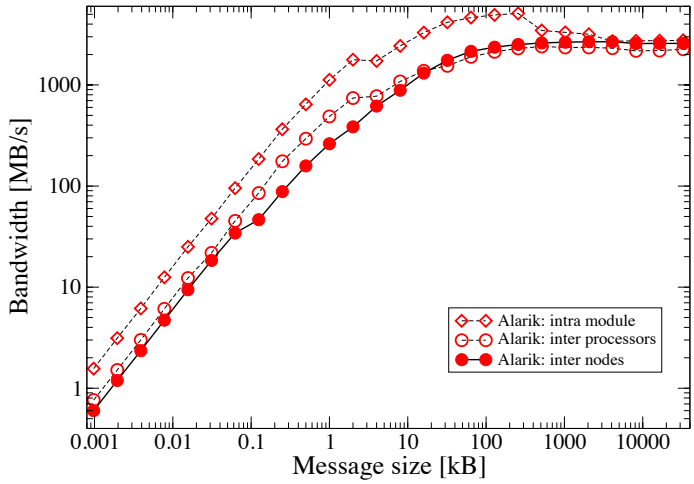

LUNARC
Streams: Loading data from Memory
Streams Triad: Total Memory bandwidth [GB/s]
0
10
20
30
40
50
60
70
80
bound sub-sequent, mem-affinity
bound spaced, mem-affinity
unbound
bound sub-sequent, mem numa-node 0
Platon
1
2
4
8
16
Active threads
Alarik: Architecture and Performance 13


LUNARC
Data exchange between cores in a module
Message passing inside a module:
• Small: L2
• Medium: L3
• Large: Main memory
Intra Module communication
Alarik: Architecture and Performance 14

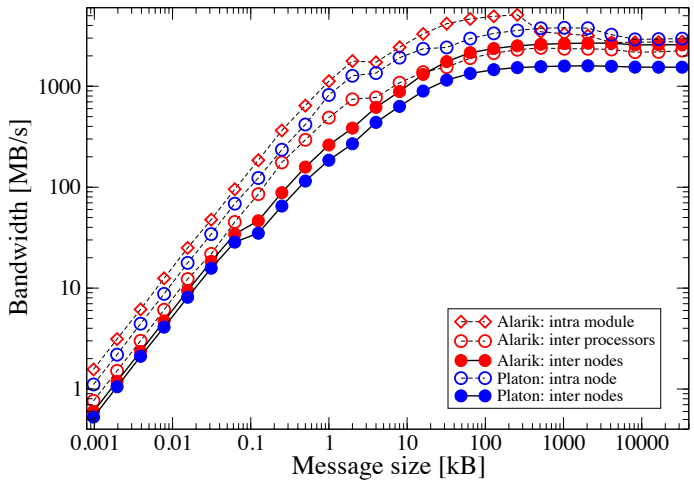

LUNARC
Data exchange between processors in a node
Message passing between processors inside a node:
• All sizes: via Hypertransport
Intra node communication
Alarik: Architecture and Performance 15

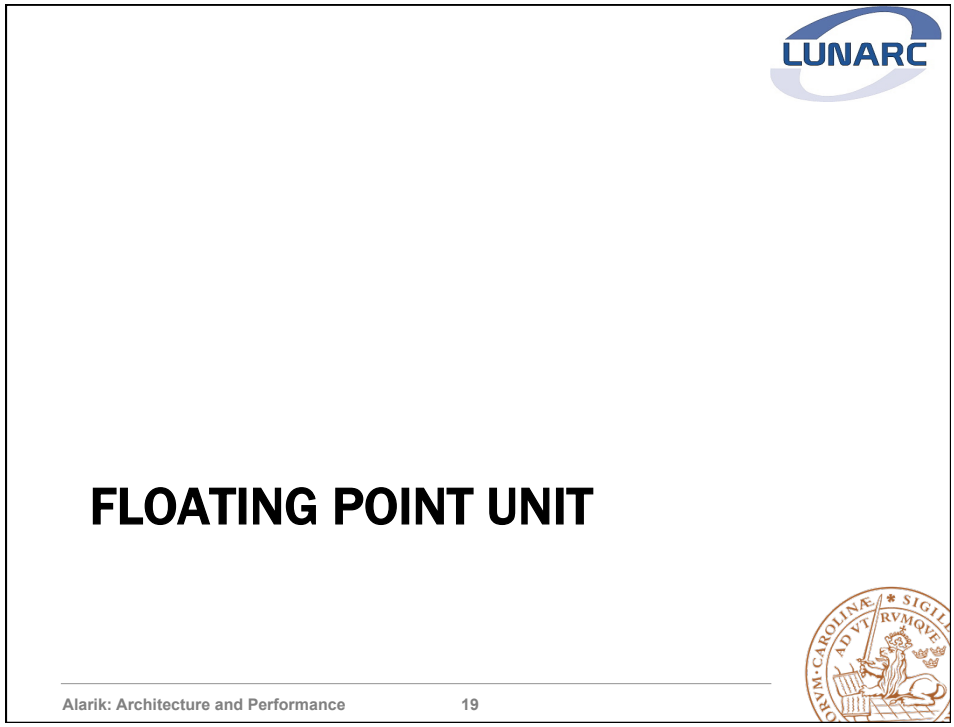

LUNARC
Data exchange between nodes
Message passing between nodes:
• All sizes: via IB-network
• RDMA, transfer doesn't involve cores
IB Switch
Inter node communication
Alarik: Architecture and Performance 16

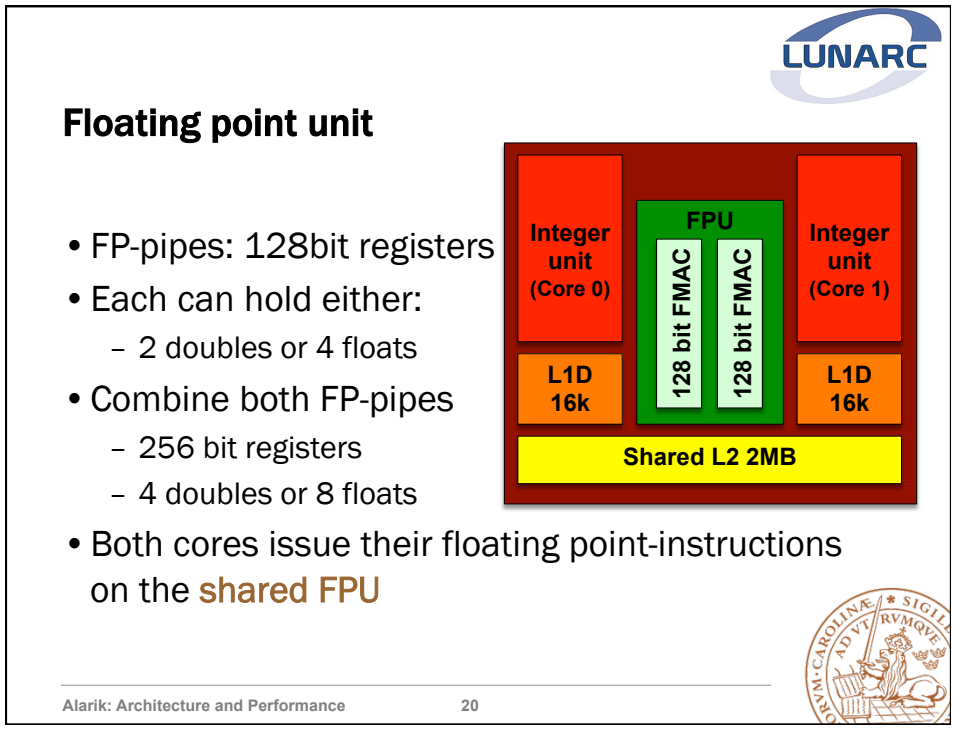
Ping-Pong bandwidth for a pair of tasks



Ping-Pong bandwidth for a pair of tasks Compare to Platon




LUNARC
FLOATING POINT UNIT
Alarik: Architecture and Performance 19


LUNARC
Floating point unit
• FP-pipes: 128bit registers
• Each can hold either:
– 2 doubles or 4 floats
• Combine both FP-pipes
– 256 bit registers
– 4 doubles or 8 floats
Integer unit (Core 0)
FPU
128 bit FMAC
128 bit FMAC
Integer unit (Core 1)
L1D 16k
L1D 16k
Shared L2 2MB
• Both cores issue their floating point-instructions on the shared FPU
Alarik: Architecture and Performance 20

Floating point FMA instructions

- Fused-Multiply-Add (FMA) aka. Fused-Multiply-Accumulate (FMAC):

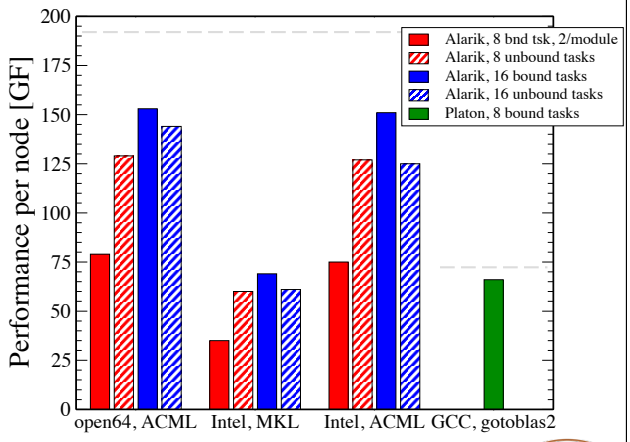
```
a = b + c * d
```

- Interlagos: FMA4 implemented - 4 different registers
- Not using FMA: half of theoretical performance lost

- SSE or AVX instructions: 4 results per cycle
- Theoretical peak of: 8 Flops per FPU per cycle
  - Shared between two cores
    - Integer, logical, address, etc. on the cores

High Performance Linpack results

- Benefit for sgl. task per module
  - private FPU
- MKL about half performance (no FMA in MKL?)
- Unbound: noisy
- Binding helps (noise & perf.) **if done right!!!**

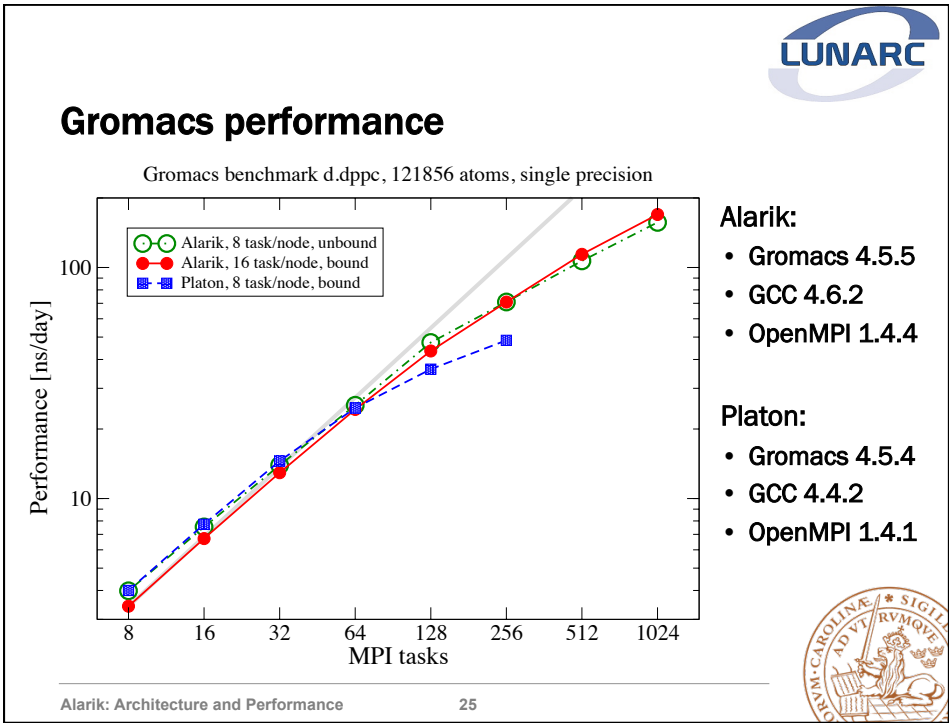


APPLICATION PERFORMANCE

Gromacs

- Molecular dynamics package
- Primarily for biochemical molecules (e.g. proteins, lipids)
- Assembler kernel (SSE instructions)

- Studying d.dppc benchmark, single precision, 122 katoms
  - Hard cut off, no PME
- GCC compile (v4.6.2)
- On 32 cores, adding:
  `-march=bdver1 –mtune=bdver1`
  boosts performance from 12.6 ns/day to 12.9 ns/day
- Disallowing AVX instructions reduces to 12.7 ns/day



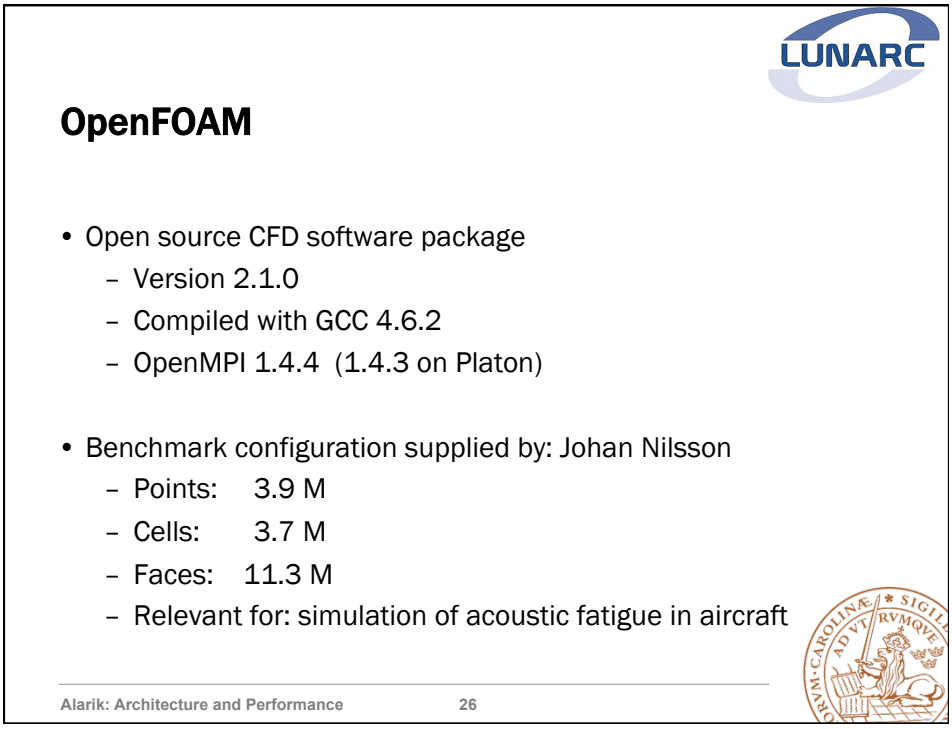
Gromacs performance

Alarik:

- Gromacs 4.5.5
- GCC 4.6.2
- OpenMPI 1.4.4

Platon:

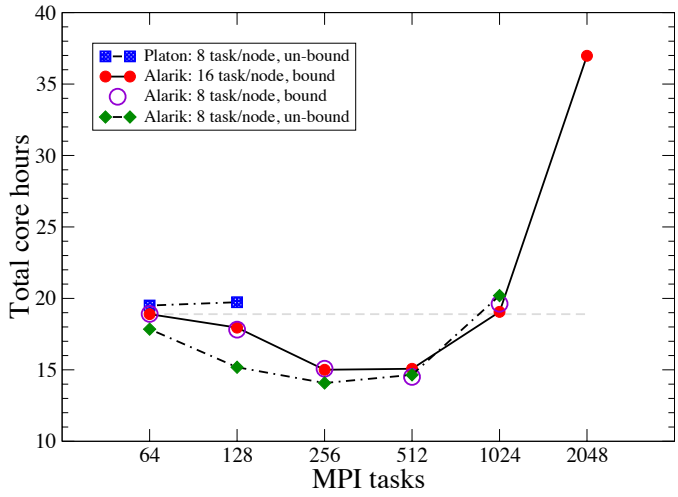
- Gromacs 4.5.4
- GCC 4.4.2
- OpenMPI 1.4.1



OpenFOAM

- Open source CFD software package
  - Version 2.1.0
  - Compiled with GCC 4.6.2
  - OpenMPI 1.4.4 (1.4.3 on Platon)
- Benchmark configuration supplied by: Johan Nilsson
  - Points: 3.9 M
  - Cells: 3.7 M
  - Faces: 11.3 M
  - Relevant for: simulation of acoustic fatigue in aircraft

OpenFOAM benchmark results: sonic



Unbound MPI codes with 16 task per node

- Unbound MPI with 16 task per node can be slow and/or noisy
- Unproblematic with undersubscribed nodes (e.g. 8 task/node)

MPI runs on 256 cores, 16 nodes

Application:	Gromacs	OpenFOAM
Bound (ref):	0.33 h/ns	211 s
Unbound: #1	13.1 h/ns	3047 s
Unbound: #2	11.3 h/ns	2819 s
Unbound: #3	11.5 h/ns	2371 s

- Use `–bind-to-core` option of `mpiexec`:

```
mpiexec –bind-to-core mdrun_mpi
```

- Don't use `–bind-to-core` for excl. 8 tsk/node

Acknowledgements

- Johan Nilsson, Construction, Lund University
- Joshua Mora, AMD
- Jakob Sandgren, Southpole AB

- The Lunarc Team
  - Jonas Lindemann
  - Magnus Ullner
  - Rickard Nilsson
  - Anders Sjöström
  - Anders Follin
  - Robert Grabowski
  - Alex Contis

Conclusions

- **Alarik is ready for service – use it for your science**

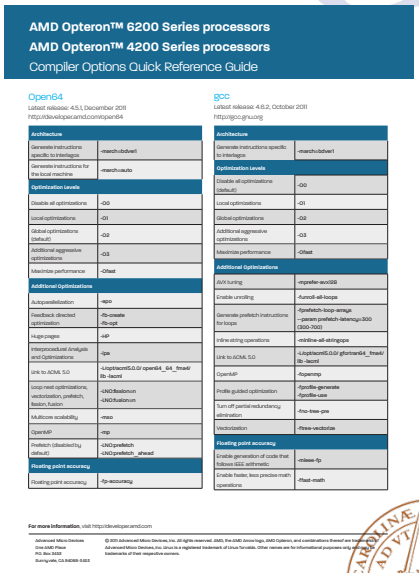
- Alarik shows very good scalability for parallel applications
  - Better scope for threaded applications

- Floating point intensive kernels should issue FMA instructions

- Performance can be boosted by placement & binding
  - Comms, Memory Bandwidth, Floating point performance
  - Often conflicting requirements
  - Understand and experiment with your application

Compilers offered

- Installed:
  - GCC 4.6.2
  - Open64 5.0.0
  - Intel 12.1
- Interlagos support: GCC & Open64
  - `-march=bdver1`
- Intel: avoid avx-inst.
  - `-msse3`



AMD Opteron™ 6200 Series processors
AMD Opteron™ 4200 Series processors
Compiler Options Quick Reference Guide

Open64
Latest release: 4.5.1, December 2011
http://developer.amd.com/open64

Architecture	
Generate instructions specific to interlagos	-march=bdver1
Generate instructions for the local machine	-march=auto
Optimization Levels	
Disable all optimizations	-O0
Local optimizations	-O1
Global optimizations (default)	-O2
Additional aggressive optimizations	-O3
Maximize performance	-Ofast
Additional Optimizations	
Autoparallelization	-apo
Feedback directed optimization	-fb-create -fb-opt
Huge pages	-HP
Interprocedural Analysis and Optimizations	-ipa
Link to ACML 5.0	-L/opt/acml5.0.0/open64_64_fma4/lib -lacml
Loop nest optimizations, vectorization, prefetch, fission, fusion	-LNO:fusion=n -LNO:fusion=n
Multicore scalability	-mso
OpenMP	-mp
Prefetch (disabled by default)	-LNO:prefetch -LNO:prefetch_ahead
Floating point accuracy	
Floating point accuracy	-fp-accuracy

gcc
Latest release: 4.6.2, October 2011
http://gcc.gnu.org

Architecture	
Generate instructions specific to interlagos	-march=bdver1
Optimization Levels	
Disable all optimizations (default)	-O0
Local optimizations	-O1
Global optimizations	-O2
Additional aggressive optimizations	-O3
Maximize performance	-Ofast
Additional Optimizations	
AVX tuning	-mprefer-avx128
Enable unrolling	-funroll-all-loops
Generate prefetch instructions for loops	-fprefetch-loop-arrays --param prefetch-latency=300 (300-700)
Inline string operations	-minline-all-stringops
Link to ACML 5.0	-L/opt/acml5.0.0/gfortran64_fma4/lib -lacml
OpenMP	-fopenmp
Profile guided optimization	-fprofile-generate -fprofile-use
Turn off partial redundancy elimination	-fno-tree-pre
Vectorization	-ftree-vectorize
Floating point accuracy	
Enable generation of code that follows IEEE arithmetic	-mieee-fp
Enable faster, less precise math operations	-ffast-math

For more information, visit http://developer.amd.com