

Towards Modeling and Simulation of Exascale Computing Platforms

Luka Stanisic

Supervised by: A.Legrand, B.Videau and J.F.Méhaut
Laboratoire d'Informatique de Grenoble
MESCAL and NANOSIM teams

June 21, 2012

Introduction

- Future super-computer platforms will be facing big challenges due to the enormous power consumption
- This internship was part of two research projects:
 - ① Mont-Blanc (European): Developing scalable and power efficient HPC platform based on low-power ARM processors
 - ② SONGS (ANR): Designing unified and open simulation framework for performance evaluation of next generation systems
- Adequate models are required
- **Goal:** Investigate is it possible to model CPU behavior at coarse grain, especially **ARM** processors

Simulation vs. alternative approach

Simulation (cycle-accurate simulation) and emulation:

- Often too slow
- Questionable accuracy

Simulation vs. alternative approach

Simulation (cycle-accurate simulation) and emulation:

- Often too slow
- Questionable accuracy

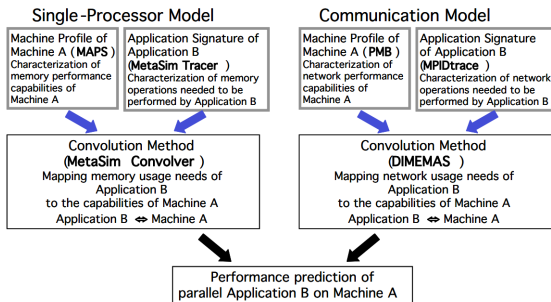
We need *coarse-grain* models:

- Lots of existing projects: LAPSE, MPI-SIM, BigSIM, MPI-NetSim, MicroGrid, **PMAC**, ...
- *Memory* is the *bottleneck* of most HPC applications
- Starting point of this work were 2 articles from Allan Snaveley and his team (PMAC) that seemed very promising:
 - 1 “A Framework for Application Performance Modeling and Prediction”. A.Snaveley, L.Carrington, N.Wolter, J.Labarta, R.Badia, A.Purkayastah, in *SuperComputing* 2002
 - 2 “A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations”. M. Tikir, L. Carrington, E. Strohmaier, A. Snaveley in *SuperComputing* 2007

Framework for Application Performance Modeling and Prediction

Authors propose a macroscopic approach:

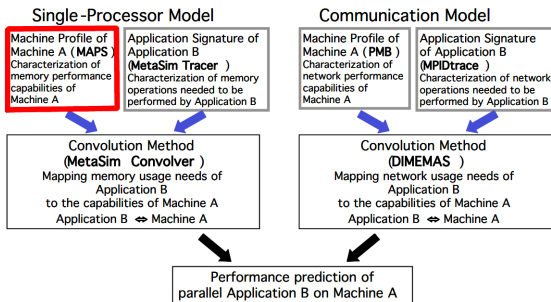
- Trying to characterize the code as a whole with parameters that can later be related to platform characteristics in order to evaluate performances



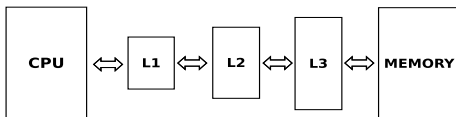
Framework for Application Performance Modeling and Prediction

Authors propose a macroscopic approach:

- Trying to characterize the code as a whole with parameters that can later be related to platform characteristics in order to evaluate performances



Kernel from MultiMAPS



MultiMAPS(size, stride, nloops)

allocate buffer *size*;

timer start;

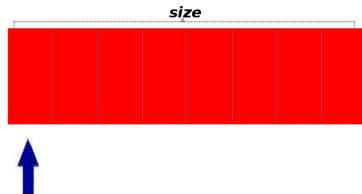
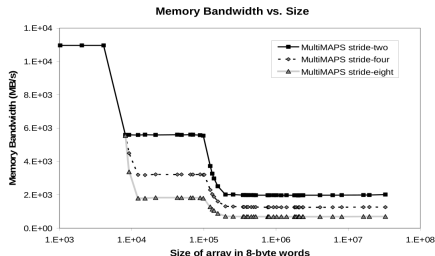
for(*i*=1:*nloops*)

access elements in buffer
by *stride*;

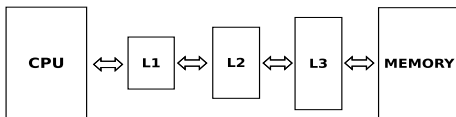
timer stop;

bandwidth=#accesses/time;

deallocate buffer;



Kernel from MultiMAPS



MultiMAPS(size, stride, nloops)

allocate buffer *size*;

timer start;

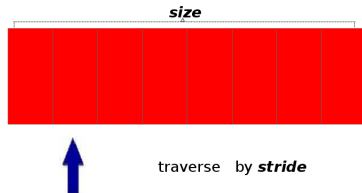
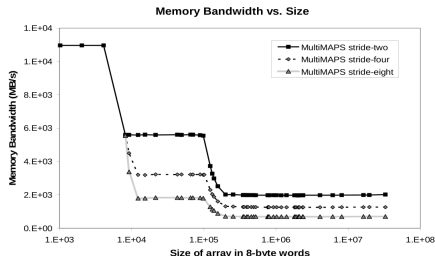
for(*i*=1:*nloops*)

access elements in buffer
by *stride*;

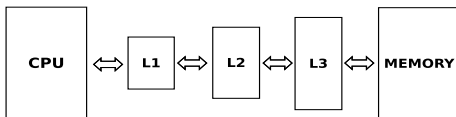
timer stop;

bandwidth=#accesses/time;

deallocate buffer;



Kernel from MultiMAPS



MultiMAPS(size, stride, nloops)

allocate buffer *size*;

timer start;

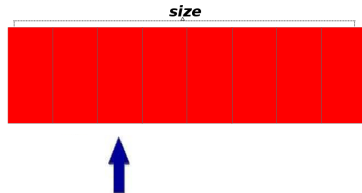
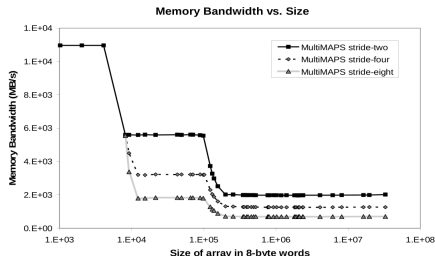
for(*i*=1:*nloops*)

 access elements in buffer
 by *stride*;

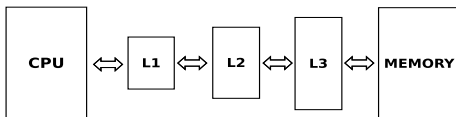
timer stop;

bandwidth=#accesses/time;

deallocate buffer;



Kernel from MultiMAPS



MultiMAPS(*size*,*stride*,*nloops*)

allocate buffer *size*;

timer start;

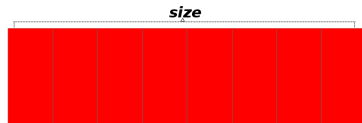
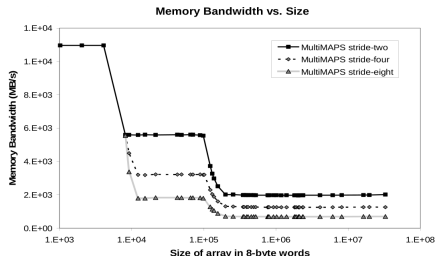
for(*i*=1:*nloops*)

 access elements in buffer
 by *stride*;

timer stop;

bandwidth=#accesses/time;

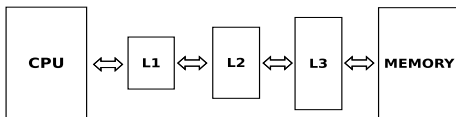
deallocate buffer;



repeating ***nloops*** times



Kernel from MultiMAPS



MultiMAPS(size, stride, nloops)

allocate buffer *size*;

timer start;

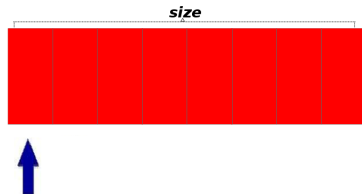
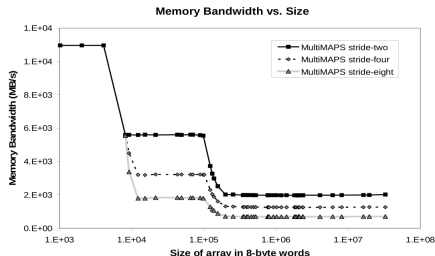
for(*i*=1:*nloops*)

 access elements in buffer
 by *stride*;

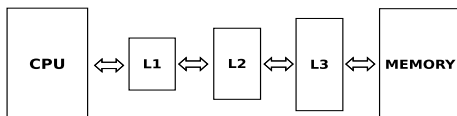
timer stop;

bandwidth=#accesses/time;

deallocate buffer;



Kernel from MultiMAPS



MultiMAPS(*size, stride, nloops*)

allocate buffer *size*;

timer start;

for(*i*=1:*nloops*)

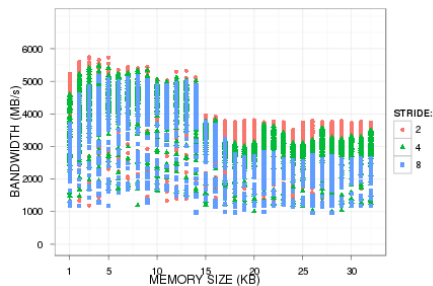
 access elements in buffer
 by *stride*;

timer stop;

bandwidth=#accesses/time;

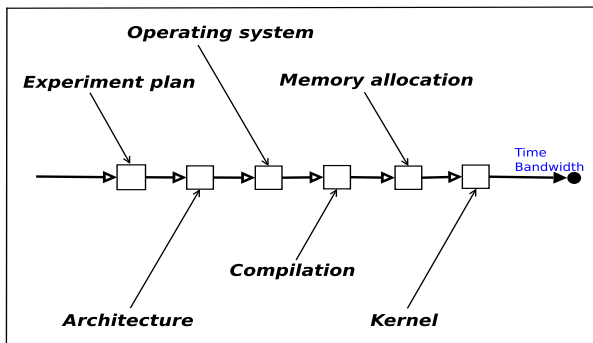
deallocate buffer;

Our first experiments:



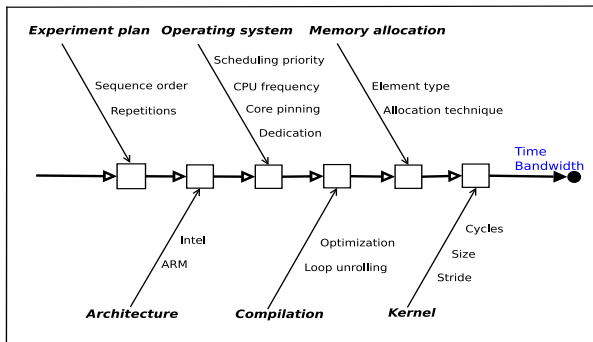
Methodology

- Problem with the related work is that it is not very well documented, it is not suited for NUMA, multicore architectures and experiments are not reproducible
- We wanted to do the measurements in a clean, coherent and systematic way



Methodology

- Problem with the related work is that it is not very well documented, it is not suited for NUMA, multicore architectures and experiments are not reproducible
- We wanted to do the measurements in a clean, coherent and systematic way

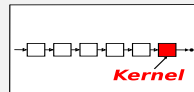


Outline

- 1 Kernel Parameters
- 2 Memory Allocation Parameters
- 3 Optimization Parameters
- 4 Operating System Parameters
- 5 Conclusion

Outline

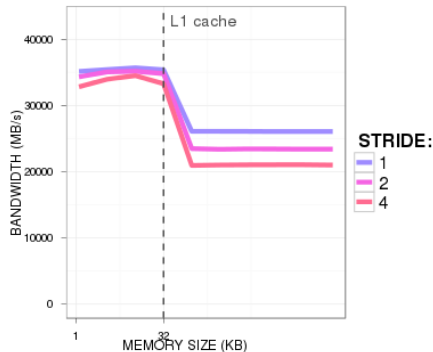
- 1 Kernel Parameters
- 2 Memory Allocation Parameters
- 3 Optimization Parameters
- 4 Operating System Parameters
- 5 Conclusion

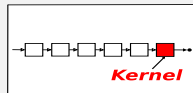


Influence of Stride Parameter

- Comparing with the results from MultiMAPS:
 - 1 Clear plateaus
 - 2 Sharp drop when getting out of the L1 cache size
 - 3 Performance is lower for larger strides

Intel Core i7 Sandy Bridge processor:
Few max values



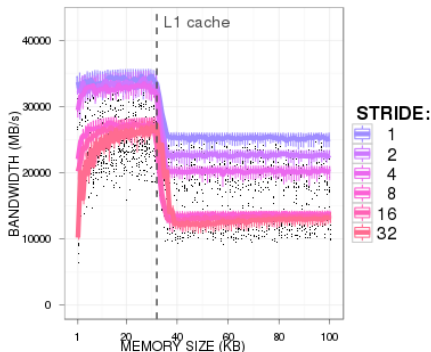


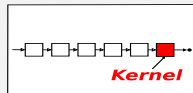
Influence of Stride Parameter

- Comparing with the results from MultiMAPS:

- 1 Clear plateaus
- 2 Sharp drop when getting out of the L1 cache size
- 3 Performance is lower for larger strides
- 4 Different bandwidths for strides 8, 16, 32 inside L1 cache size
- 5 Performance drop for higher memory size values stop after stride 8

Intel Core i7 Sandy Bridge processor: Randomization + Boxplots





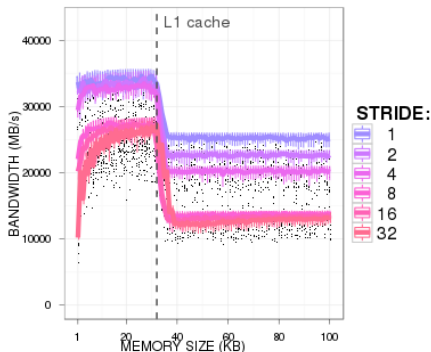
Influence of Stride Parameter

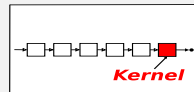
- Comparing with the results from MultiMAPS:

- 1 Clear plateaus
- 2 Sharp drop when getting out of the L1 cache size
- 3 Performance is lower for larger strides
- 4 Different bandwidths for strides 8, 16, 32 inside L1 cache size
- 5 Performance drop for higher memory size values stop after stride 8

- This is general behavior, but with many exceptions

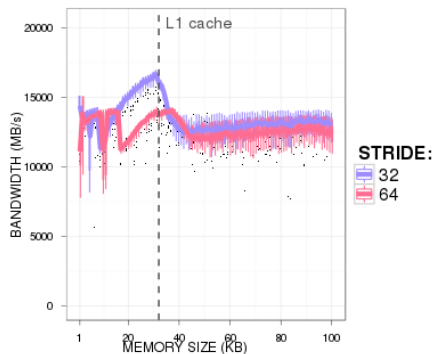
Intel Core i7 Sandy Bridge processor: Randomization + Boxplots



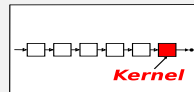


Unexpected Behavior

Example for Intel Core i7 3.40 GHz
Sandy Bridge:

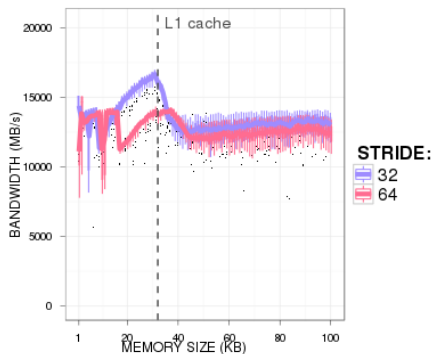


- Irregular behavior inside L1 cache size!

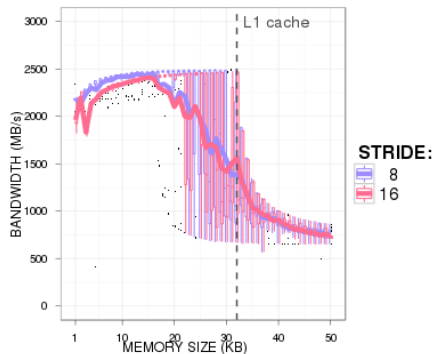


Unexpected Behavior

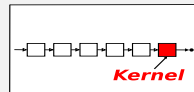
Example for Intel Core i7 3.40 GHz
Sandy Bridge:



Example for ARM Dual Cortex A9
1 GHz Snowball:

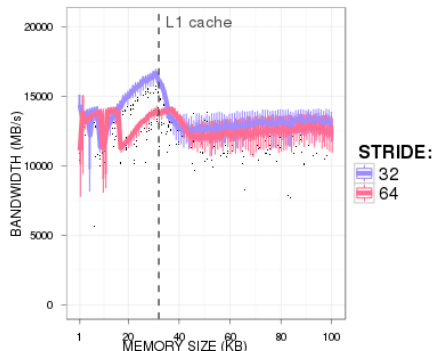


- Irregular behavior inside L1 cache size!



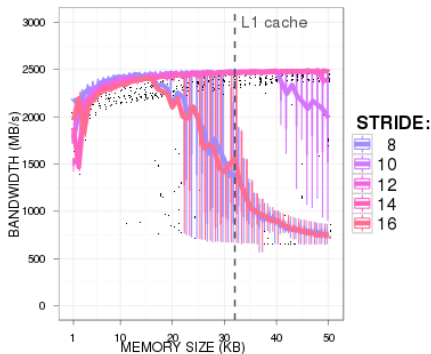
Unexpected Behavior

Example for Intel Core i7 3.40 GHz
Sandy Bridge:



- Irregular behavior inside L1 cache size!

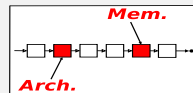
Example for ARM Dual Cortex A9
1 GHz Snowball:



- Strides 10, 12, 14 have better performance than Stride 8 !?!

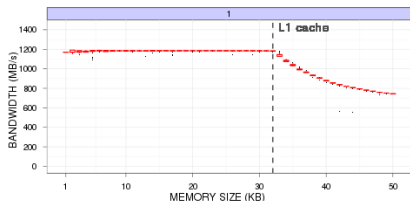
Outline

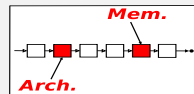
- 1 Kernel Parameters
- 2 Memory Allocation Parameters**
- 3 Optimization Parameters
- 4 Operating System Parameters
- 5 Conclusion



Reproducibility Issue on ARM

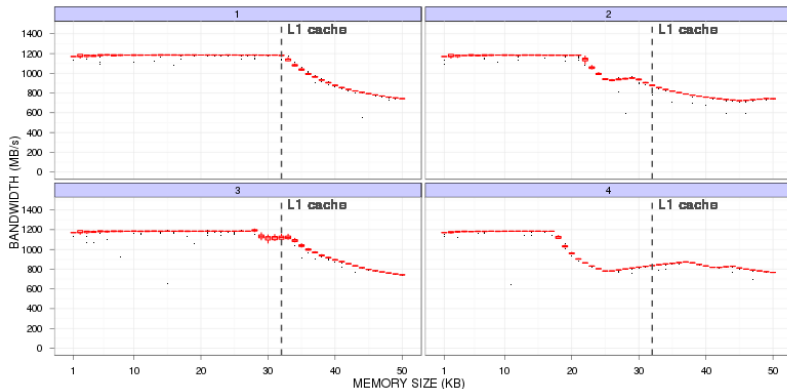
- Same input parameters, consecutive experiments
- 42 repetitions per each memory size, NO NOISE!
- Results from ARM Dual Cortex A9 1GHz (Snowball):

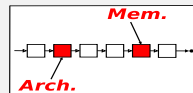




Reproducibility Issue on ARM

- Same input parameters, consecutive experiments
- 42 repetitions per each memory size, NO NOISE!
- Results from ARM Dual Cortex A9 1GHz (Snowball):



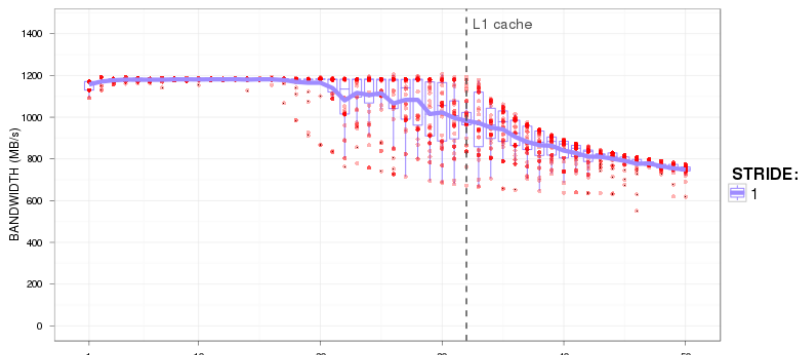


Influence of Allocation Strategy on ARM



Different memory allocation technique:

Performance depend on actual physical address:



Outline

- 1 Kernel Parameters
- 2 Memory Allocation Parameters
- 3 Optimization Parameters**
- 4 Operating System Parameters
- 5 Conclusion

Influence of Code Optimizations

Element type

Using *long long int* which is 64b instead of regular *int* 32b

Vectorized instructions:

- On Intel: 128b SSE and 256b AVX
- On ARM: 128b NEON

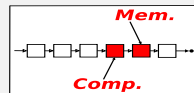
Loop unrolling

Standard execution:

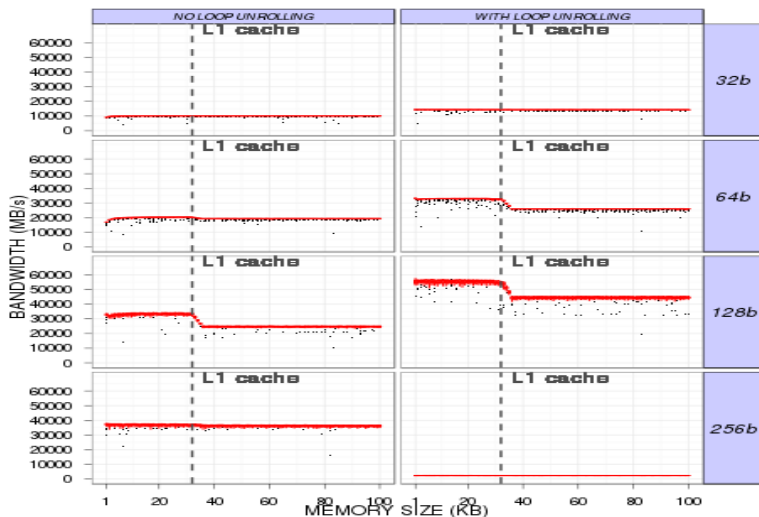
```
for(j=0;j<bufferSize;j+=STRIDE)
{
    sum+=buffer[j];
}
```

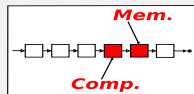
With loop unrolling:

```
for(j=0;j<bufferSize;j+=STRIDE*8)
{
    sum+=buffer[j];
    ...
    sum+=buffer[j+7*STRIDE];
}
```

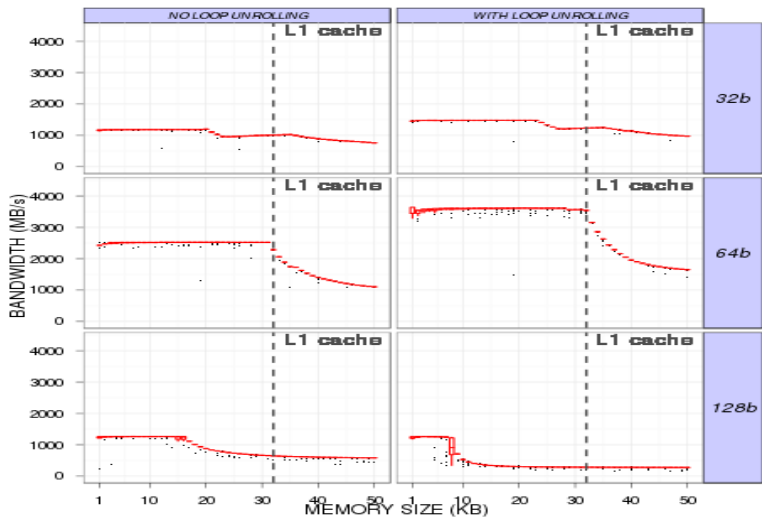


Results from Intel Sandy Bridge:



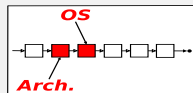


Results from ARM Snowball:



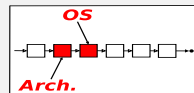
Outline

- 1 Kernel Parameters
- 2 Memory Allocation Parameters
- 3 Optimization Parameters
- 4 Operating System Parameters**
- 5 Conclusion



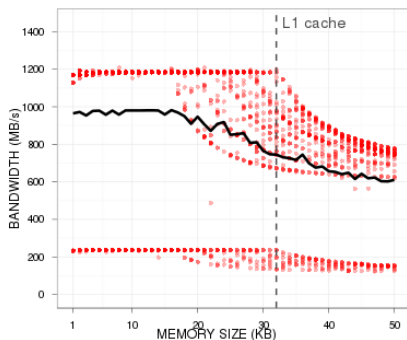
Influence of OS Scheduling Policy on ARM

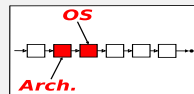
- ① **Default** priority-results shown on previous slides
- ② **Nice** priority-same behavior as default priority
- ③ **Real-time** priority-distinctive output



Influence of OS Scheduling Policy on ARM

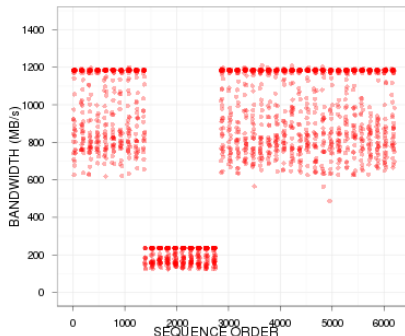
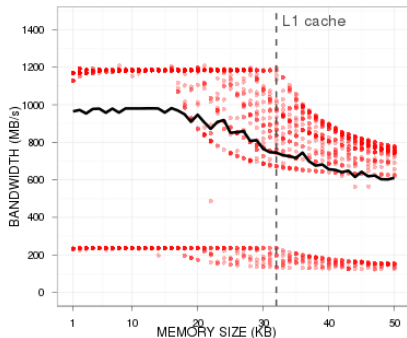
- ① Default priority-results shown on previous slides
- ② Nice priority-same behavior as default priority
- ③ **Real-time** priority-distinctive output
 - Demonstration of 2 modes of execution for real-time priority:





Influence of OS Scheduling Policy on ARM

- ① Default priority-results shown on previous slides
- ② Nice priority-same behavior as default priority
- ③ Real-time priority-distinctive output
 - Demonstration of 2 modes of execution for real-time priority:



Outline

- 1 Kernel Parameters
- 2 Memory Allocation Parameters
- 3 Optimization Parameters
- 4 Operating System Parameters
- 5 Conclusion

Conclusion

- This research provides insights on the possible use of ARM processors in HPC
- Predicting memory behavior is harder than the literature suggests
 - ① Many unexpected parameters have great influence
 - ② ARM processors should have been simpler but they are not (e.g. physical address on ARM)
 - ③ Optimized versions are generally more regular and easier to model
- Finding the good factor combination is non trivial
- Open Science: organize so that anyone can check and easily reproduce!

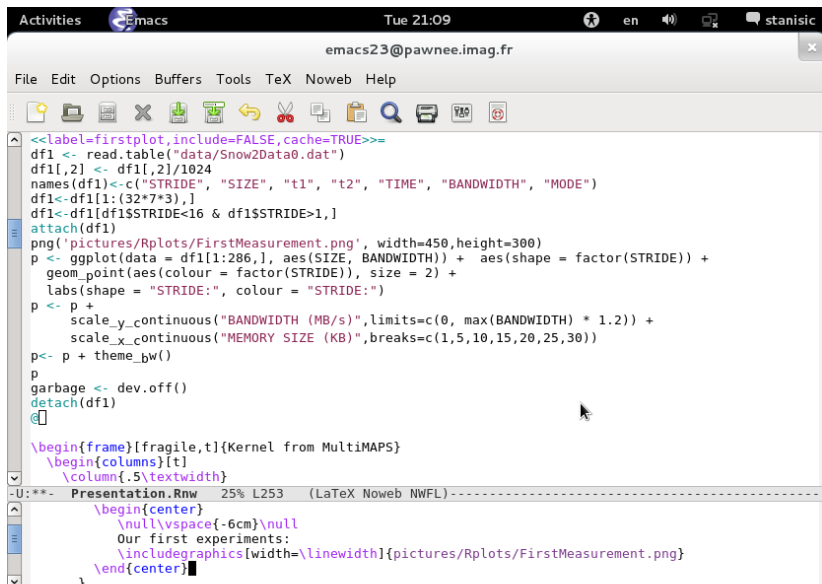
Future work

- Investigate more elaborate kernels that put more pressure on CPU
- Multi-threaded version of kernel (share data): synchronization protocol overhead, hierarchy
- Different kernels on the same node and sharing cache hierarchy:

Future work

- Investigate more elaborate kernels that put more pressure on CPU
 - Multi-threaded version of kernel (share data): synchronization protocol overhead, hierarchy
 - Different kernels on the same node and sharing cache hierarchy:
-
- Incorporate HPC network models, GPU models, ...
 - Use these models with simulation tools (SimGrid) to predict performance of future computer platforms
 - Try to use open-science/reproducible research techniques in the HPC field and advertise for it

Sweave + Beamer



```
Activities Emacs Tue 21:09 en stanisic
emacs23@pawnee.imag.fr

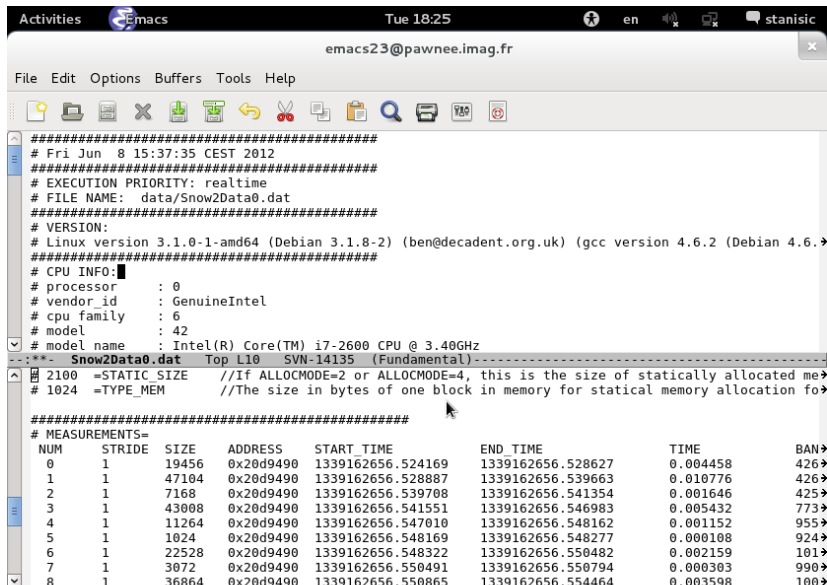
File Edit Options Buffers Tools TeX Noweb Help

<<label=firstplot,include=FALSE,cache=TRUE>>=
df1 <- read.table("data/Snow2Data0.dat")
df1[,2] <- df1[,2]/1024
names(df1)<-c("STRIDE", "SIZE", "t1", "t2", "TIME", "BANDWIDTH", "MODE")
df1<-df1[1:(32*7*3),]
df1<-df1[df1$STRIDE<16 & df1$STRIDE>1,]
attach(df1)
png('pictures/Rplots/FirstMeasurement.png', width=450,height=300)
p <- ggplot(data = df1[1:286,], aes(SIZE, BANDWIDTH)) + aes(shape = factor(STRIDE)) +
  geom_point(aes(colour = factor(STRIDE)), size = 2) +
  labs(shape = "STRIDE:", colour = "STRIDE:")
p <- p +
  scale_y_continuous("BANDWIDTH (MB/s)",limits=c(0, max(BANDWIDTH) * 1.2)) +
  scale_x_continuous("MEMORY SIZE (KB)",breaks=c(1,5,10,15,20,25,30))
p<- p + theme_bw()
p
garbage <- dev.off()
detach(df1)
@

\begin{frame}[fragile,t]{Kernel from MultiMAPS}
\begin{columns}[t]
\column{.5\textwidth}

-U:**- Presentation.Rnw 25% L253 (LaTeX Noweb NWFL) -----
\begin{center}
\null\vspace{-6cm}\null
Our first experiments:
\includegraphics[width=\linewidth]{pictures/Rplots/FirstMeasurement.png}
\end{center}
```

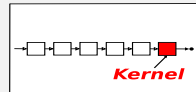
Data file



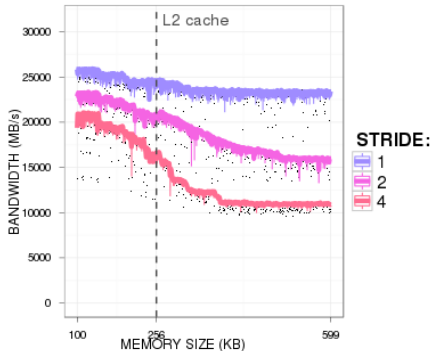
```
#####
# Fri Jun  8 15:37:35 CEST 2012
#####
# EXECUTION PRIORITY: realtime
# FILE NAME:  data/Snow2Data0.dat
#####
# VERSION:
# Linux version 3.1.0-1-amd64 (Debian 3.1.8-2) (ben@decadent.org.uk) (gcc version 4.6.2 (Debian 4.6.2)
#####
# CPU INFO:
# processor      : 0
# vendor_id     : GenuineIntel
# cpu family    : 6
# model        : 42
# model name    : Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
--:**- Snow2Data0.dat  Top L10  SVN-14135  (Fundamental)-----
# 2100 =STATIC_SIZE //If ALLOCMODE=2 or ALLOCMODE=4, this is the size of statically allocated me
# 1024 =TYPE_MEM    //The size in bytes of one block in memory for statical memory allocation fo

#####
# MEASUREMENTS=
# NUM STRIDE SIZE ADDRESS START_TIME END TIME TIME BAN
0 1 19456 0x20d9490 1339162656.524169 1339162656.528627 0.004458 426
1 1 47104 0x20d9490 1339162656.528887 1339162656.539663 0.010776 426
2 1 7168 0x20d9490 1339162656.539708 1339162656.541354 0.001646 425
3 1 43008 0x20d9490 1339162656.541551 1339162656.546983 0.005432 773
4 1 11264 0x20d9490 1339162656.547010 1339162656.548162 0.001152 955
5 1 1024 0x20d9490 1339162656.548169 1339162656.548277 0.000108 924
6 1 22528 0x20d9490 1339162656.548322 1339162656.550482 0.002159 101
7 1 3072 0x20d9490 1339162656.550491 1339162656.550794 0.000303 990
8 1 36864 0x20d9490 1339162656.550865 1339162656.554464 0.003598 100
```

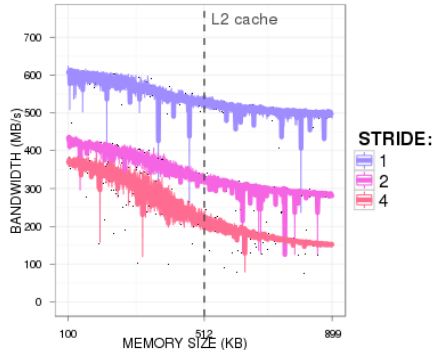

Large Buffer Size



Example for Intel Core i7
3.40GHz Sandy

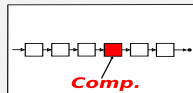


Example for ARM Dual Cortex
A9 1GHz Bridge: Snowball:

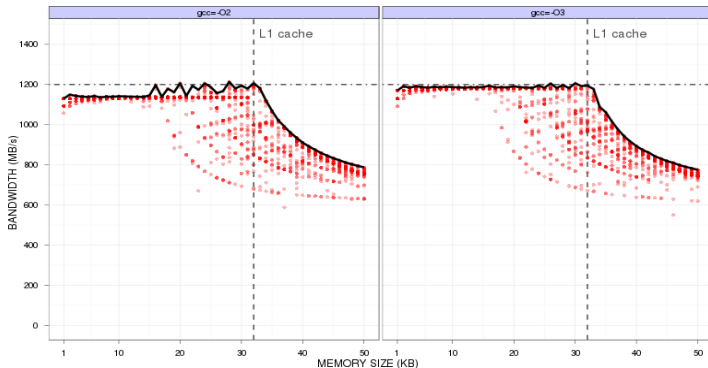


The drop around L2 is smooth,
no sharp plateaus

Influence of Compiler Optimization Option



Using different compilation optimizations affect the performance
Results from ARM Dual Cortex A9 1GHz Snowball:



Intel processors also show better performance with gcc=-O3