http://www.montblanc-project.eu

Using the MontBlanc prototype for the solution of large sparse linear systems

Riccardo Rossi, Pooyan Dadvand
CIMNE (NUMEXAS)

Denis Demidov

Supercomputer Center of the Russian Academy of Sciences

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 288777.

Tested application

AMGCL

Algebraic MultiGrid preconditioner + Deflation for the scalable solution of large system of equations

Programming language(s)

C, C++, Python, OpenCL

Parallel programming model(s)

MPI, OpenMP,

Accelerator programming model(s)

OpenCL, …

Libraries

Pastix, BLAS

Resources required

- Memory requirement per node (GB) – **no specific requirements** (more nodes needed for larger problem sizes)
- Storage required – **no specific requirements**
- Minimum number of nodes / CPU - **1**

Problem Description

- The target problem is the solution of a linear system of equations of the type: (large & sparse!)

$$A\,x = b$$

- Our effort is to implement a scalable solver which combines a **local Algebraic MultiGrid preconditioner** with a "**deflation technique**" so to achieve an algorithmically scalable solution procedure.

Algorithmic Approach - Deflation

- Let's consider a problem of size “n” to be run on “N” processors.

- The idea is that the solution “x” is divided in a coarse solution and a correction as

$$x = x_{coarse} + y$$

- xcoarse is then expressed as:

$$x_{coarse} = W\, z$$

- where “W” is a rectangular matrix of size n by N, whose columns are linearly independent one from the others (details on the construction of W are out of scope here)

Algorithmic Approach - Deflation

- The idea is to determine x_{coarse} and y separately, by solving first a “coarse” problem of the type

$$W^T A W z = W^T b$$

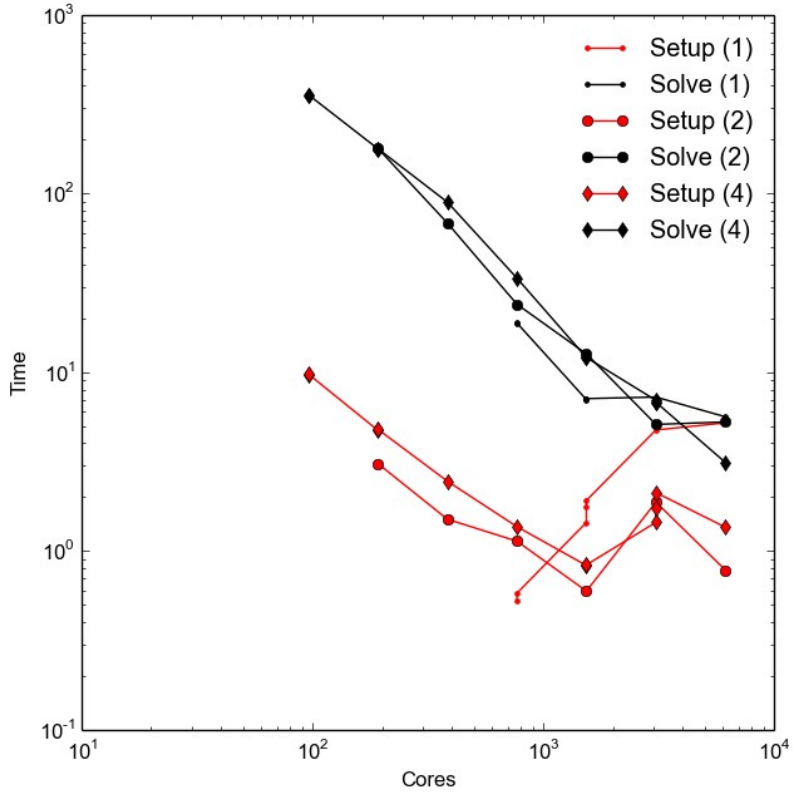
- and then by determining the correction “y” (A-orthogonal to the coarse solution) which solves the original problem

ADVANTAGES:

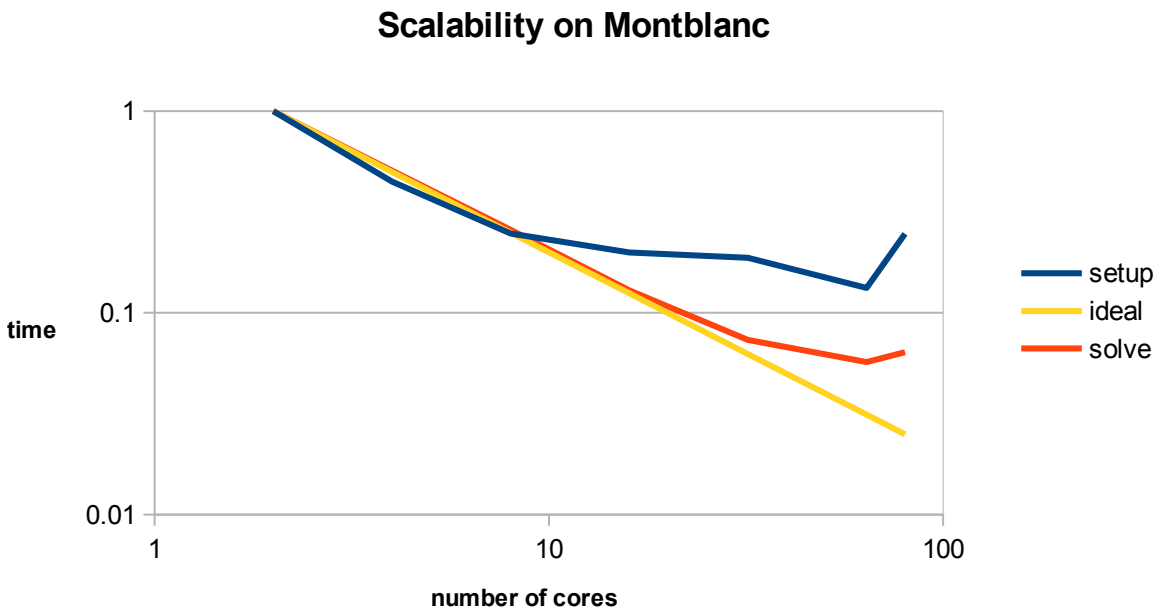
- Number of iterations for the solution of the linear system by a Krylov method DIMINISHES with the number of MPI processes
- Can be combined with an UNMODIFIED single field preconditioner (AMGCL in our case)

Scalability (on a Cray XC30)

On a “standard” x86 system (CRAY XC30) the solver has proved good strong scaling properties



MPI Scalability on Montblanc



Arithmetic intensity and communications are both higher in setup phase.
Still scalability should be close to perfect with this number of processors

SMP parallelism – Comparison to others

The problem used for benchmarking is 3D Poisson problem in a unit cube with Dirichlet boundary conditions. The grid size is 100x100x100 nodes (1e6 unknowns).

	Setup (sec)	Solution (sec)	Iterations
CPU	2.480	9.090	16
GPU	3.816	23.558	15

Same results for a Tesla K40c GPU and an Intel Core i7 930 CPU:

	Setup (sec)	Solution (sec)	Iterations
CPU	0.773	2.439	16
GPU	1.263	0.484	16

CPU is 3-4 times slower than I7. GPU is not really fast either

Some results of VEXCL – on Montblanc

Double precision

Test	GPU GFLOPS	GPU GB/sec	CPU GFLOPS	CPU GB/sec
SAXPY	0.447168	5.36601	0.317687	3.81224
Reduction	0.675904	5.40723	0.411466	3.29173
Convolution	0.592428	4.73943	0.411426	3.29141
SpMV	0.392066	4.51773	0.30536	3.51863

Single precision

Test	GPU GFLOPS	GPU GB/sec	CPU GFLOPS	CPU GB/sec
SAXPY	0.689316	4.13589	0.590263	3.54158
Reduction	0.982317	3.92927	0.657627	2.63051
Convolution	0.914395	3.65758	0.417853	1.67141
SpMV	0.434536	2.90801	0.367598	2.46005

VEXCL: https://github.com/ddemidov/vexcl

Some results of VEXCL – on X86

Test	GPU GFLOPS	GPU GB/sec	CPU GFLOPS	CPU GB/sec
SAXPY	16.9368	203.241	0.923484	11.0818
Reduction	27.4459	219.567	0.987185	7.89748
Convolution	138.411	1107.29	0.953322	7.62658
SpMV	28.8368	385.965	0.977774	13.087

Reference results on I7 930 + Tesla K40c

CPU is only twice faster, but GPU really shines!

Feedback:

“minor” things:

- Had to replace v1.2 of CL/cl.hpp (distributed by Khronos) with v1.0. Otherwise got compilation errors listed below. It would be a good idea to provide the compatible CL/cl.hpp at system level on Mont-Blanc prototype.

CL/cl.hpp: In function ‘void cl::detail::fence()’:
CL/cl.hpp:1041:38: error: ‘_mm_mfence’ was not declared in this scope

- Had to ignore the fact that cl_khr_fp64 extension was not listed as available. Double precision still worked.
- Had to explicitly link to /apps/opencl_sdk/1.1.0/lib/libOpenCL.so. The default /usr/local/lib/libOpenCL.so that was found by cmake did not work.
- As a curiosity: overflow behavior for doubles appear to be slightly different on ARM and X86

Acknowledgments:

Dr. Rossi and Dr. Dadvand would like to acknowledge the support of the European Commission through the NUMEXAS project - grant n° 611636

Dr. Demidov acknowledges the support of the the Russian Foundation for Basic Research (RFBR) grants No 12-07-0007 and 12-01-00333a