Solving sparse linear systems in a many-core environment

Mikko Byckling
Special thanks: Peter Råback

CSC - IT Center for Science Ltd., Espoo, Finland.
(mikko.byckling@csc.fi)

Sparse Days, 17th of June, 2013

Contents

Motivation I

In a modern supercomputer, a single node has several cores with shared memory. In addition, many-core coprocessors have recently appeared to complement the traditional CPUs.

Q: Are sparse flops free?

Is it already possible to leverage the available processing power when solving *sparse* linear systems?

Motivation II

In this talk, we study the scalability and efficiency of

- sparse direct methods
- iterative methods

in a purely shared memory environment.

Main purpose is to gain insight into the implementation of iterative methods in a multi/many-core environment.

Xeon

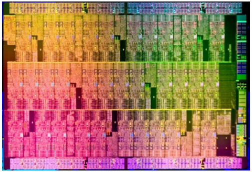
Typically each node in a supercomputer has CPU's with a shared address space.

Technical specifications for Xeon E5 2670

- Up to 8 full cores with 2-way HT (X86)
- Frequency 2.6Ghz (up to 3.3Ghz with turbo-mode)
- 256-bit SIMD unit per core
- Typically at least 2Gb of DDR3 memory available per core
- 32kb of L1 cache per core
- 256kB of L2 cache per core
- 20Mb of shared L3 cache per die
- $\sim$0.15Tflops in DGEMM
- TDP 130 Watts

Xeon Phi I

intel inside™
Xeon® Phi™



- Officially launched at SC12
- Standard software stack available (C/C++, Fortran, etc.)
- Single Xeon Phi core *much* slower than a regular Xeon core
  - Use of SIMD vectorization necessary for performance
  - Use of parallelism necessary for performance
- Porting code is very easy, getting performance is not

Xeon Phi II

Technical specifications

- Separate accelerator card
- Local host OS with Linux
- Up to 61 stripped down cores with 4-way HT (X86)
- Frequency 1.26Ghz
- 512-bit SIMD unit per core
- Up to 8Gb of GDDR5 memory
- 32kb of L1 cache per core (8-way associative)
- 512kB of L2 cache per core (8-way associative)
- FMA and gather/scatter support
- $\sim$0.83Tflops in DGEMM
- TDP 300 Watts

Test setup

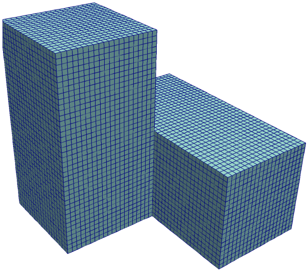
- Direct method: Intel MKL Pardiso
- Iterative method: Conjugate gradient (without preconditioning)

Sparse matrix-vector products from MKL.

Two different test problems with three different mesh sizes.

Test problem	n	$\mathrm{nz}(A)$	$\rho = \mathrm{nz}(A)/n$
3D_heat_small	35721	896761	25.1
3D_heat_med	105300	2702656	25.6
3D_heat_large	291060	7580368	26.0
3D_elas_small	24843	1786545	71.9
3D_elas_med	107163	8070849	75.3
3D_elas_large	315900	24323904	77.0

Domain: two intersecting square pipes in 3D



Set domain as two intersecting square pipes in an L-shape, with both free ends of the pipes, denoted by Γ_1 and Γ_2, having non-trivial boundary conditions.

Discretization with multiphysical FE code Elmer [1].

[1] http://www.csc.fi/english/pages/elmer

Test problem 1: Heat equation (heat) I

Consider the heat equation in a steady state with constant coefficients

$$\begin{aligned} -\Delta u &= f, && \text{in } \Omega \\ u &= 0, && \text{on } \Gamma_1 \text{ and } \Gamma_2 \\ \frac{\delta u}{\delta n} &= 0, && \text{otherwise} \end{aligned} \tag{1}$$

Let the heat loading of the system to be constant $f = 1$.

The resulting system is symmetric positive definite.

Test problem 1: Heat equation (heat) II

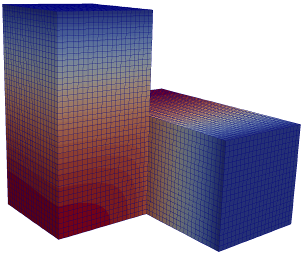


Figure: Solution of the steady-state heat equation with $h = 0.05$

Test problem 2: Linear elasticity (elas) I

Consider the linear elasticity equation with isotropic material

$$\begin{aligned} \rho\frac{\delta^2 \bar{d}}{\delta t^2} - \nabla\tau &= \bar{f}, && \text{in } \Omega \\ \bar{d} &= \bar{0}, && \text{on } \Gamma_1 \\ \tau\bar{n} &= \bar{g}, && \text{on } \Gamma_2 \\ \tau\bar{n} &= 0, && \text{otherwise,} \end{aligned} \tag{2}$$

where ρ is the density and the stress tensor τ is given as

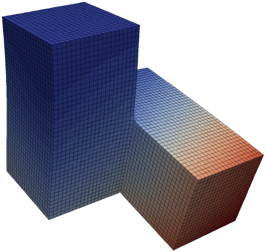
$$\tau = 2\mu\epsilon + \lambda\nabla\cdot\bar{d}I,$$

where the Lame parameters μ and λ are defined in terms of Youngs modulus Y and Poisson ratio κ as $\lambda = \frac{Y\kappa}{(1+\kappa)(1-2\kappa)}$, $\mu = \frac{Y}{2(1+\kappa)}$.

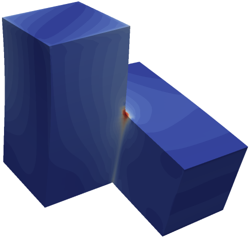
Let the system be free of volume loading $\bar{f} = 0$ and set $\rho = 1e3$, $Y = 1e9$, $\kappa = 0.2$ and $\bar{g} = (0, 1e6, 0)^T$.

The resulting system is symmetric positive definite.

Test problem 2: Linear elasticity (elas) II



(a) Displacement



(b) Tensile stress

Figure: Solution of the linear elasticity equation with $h = 0.05$.

CSC

Direct method, results

Xeon, Intel MKL Pardiso, `KMP_AFFINITY=granularity=fine,compact`

Test problem	Time (s) nt=1	nt=16	nt=32	Speedup, T_{nt}/T_1 nt=16	nt=32
3D_heat_small	0.99	**0.67**	**0.67**	1.47	1.47
3D_heat_med	5.34	**2.25**	**2.24**	2.38	2.38
3D_heat_large	30.63	**8.84**	**8.89**	3.47	3.45
3D_elas_small	1.06	**0.61**	**0.59**	1.73	1.81
3D_elas_med	12.64	**3.81**	**3.83**	3.32	3.30
3D_elas_large	92.21	19.00	**18.89**	4.85	4.88

Xeon Phi, Intel MKL Pardiso, `KMP_AFFINITY=granularity=fine,balanced`

Test problem	Time (s) nt=1	nt=60	nt=120	Speedup, T_{nt}/T_1 nt=60	nt=120
3D_heat_small	10.93	**3.82**	4.38	2.86	2.50
3D_heat_med	67.86	**12.36**	12.90	5.49	5.26
3D_heat_large	429.55	49.02	**48.23**	8.76	8.91
3D_elas_small	12.92	**3.14**	3.50	4.12	3.70
3D_elas_med	169.91	**18.63**	19.28	9.12	8.81
3D_elas_large	1492.79	86.46	**79.28**	17.26	18.83

$\Longrightarrow$ As expected, not enough parallelism for 240 threads.

NUMA aware sparse matrix-vector products I

Assumption: Sparse matrix stored in CRS format. Symmetry *not* taken into account.

Consider sparse matrix-vector product $y = Ax$, where $A \in \mathbb{R}^{n \times n}$ is large and sparse.

Flops to byte ratio is approximately $O(\frac{1}{6})$ (memory bound operation).

In the FE code Elmer used to generate the test problems, in a distributed memory setting sparse matrix vector products may take up to 80% of the total runtime.

NOTE: In CRS format, starting addresses of the rows are 64-byte *unaligned* in general, i.e., the compiler has to generate peel loops in order to vectorize the computation of the dot products.

NUMA aware sparse matrix-vector products II

Distributing the original matrix rowwise among threads, we have $y = Ax$ as

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{bmatrix}, \quad y = Ax \Leftrightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} A_1 x \\ A_2 x \\ \vdots \\ A_k x \end{bmatrix},$$

where k denotes the maximum number of threads, A_j denotes a thread local submatrix held by thread j and y_j the local part of the result vector.

Sparse matrix-vector products, Xeon

Xeon, time to compute 100 repetitions of $y = Ax$.

Test problem	Time (s) nt=1	nt=16	nt=32	Speedup, T_{nt}/T_1 nt=16	nt=32
CRS					
3D_heat_small	0.11	**0.02**	**0.02**	5.75	4.63
3D_heat_med	0.43	**0.09**	**0.09**	4.79	4.84
3D_heat_large	1.20	**0.27**	**0.27**	4.46	4.43
3D_elas_small	0.22	**0.04**	**0.04**	5.82	5.92
3D_elas_med	1.05	**0.27**	**0.27**	3.98	3.93
3D_elas_large	3.20	**0.78**	**0.78**	4.12	4.11
NUMA CRS					
3D_heat_small	0.11	**0.03**	0.05	3.96	2.33
3D_heat_med	0.39	0.10	**0.04**	4.08	9.99
3D_heat_large	1.06	0.29	**0.16**	3.72	6.73
3D_elas_small	0.19	0.04	**0.08**	4.55	2.49
3D_elas_med	0.94	0.27	**0.17**	3.47	5.43
3D_elas_large	2.73	0.78	**0.42**	3.50	6.53

Sparse matrix-vector products, Xeon Phi

Xeon Phi, time to compute 100 repetitions of $y = Ax$.

Test problem	Time (s) nt=1	nt=60	nt=240	Speedup, T_{nt}/T_1 nt=60	nt=240
CRS					
3D_heat_small	1.25	**0.16**	0.32	8.00	3.93
3D_heat_med	3.57	**0.32**	0.33	11.10	10.70
3D_heat_large	9.89	0.64	**0.55**	15.39	18.09
3D_elas_small	1.90	**0.17**	0.29	10.91	6.48
3D_elas_med	8.06	**0.38**	0.43	21.42	18.89
3D_elas_large	23.92	1.01	**0.69**	23.70	34.69
NUMA CRS					
3D_heat_small	1.22	0.26	**0.07**	4.70	16.33
3D_heat_med	3.62	0.36	**0.12**	9.97	30.81
3D_heat_large	9.89	0.26	**0.19**	38.75	52.54
3D_elas_small	1.80	0.25	**0.08**	7.16	22.86
3D_elas_med	8.03	0.20	**0.14**	39.26	56.62
3D_elas_large	23.87	0.53	**0.28**	45.37	85.82

NUMA aware conjugate gradient iteration I

Avoid false sharing and enable better vectorization by distributing the rows of matrix A and work vectors in blocks of size 64 bytes. Use `!DEC$ ASSUME_ALIGNED` and `!DEC$ SIMD` pragmas to inform the compiler.

Avoid thread join/fork overhead by letting each thread to operate only on predefined section of x, q, z and r.

Use local variables and OpenMP barriers to implement reductions and thread synchronization.

NUMA aware conjugate gradient iteration II

Preconditioned conjugate gradient algorithm

for all threads in parallel **do**
  reduce bnrm2 $\leftarrow_{+} ||b_{I:t}||_2^2$
  $r = b - Ax$
  barrier
  reduce $||r||_2^2 \leftarrow_{+} ||r_{I:t}||_2^2$
  $\text{resnorm}_L = \sqrt{||r||_2^2/\text{bnrm2}}$
  if $\text{resnorm}_L < tol$ **stop**
  for $t = 1, \ldots,$ maxit **do**
    Solve $Mz = r$
    barrier
    reduce $\rho \leftarrow_{+} r_{I:t}^T z_{I:t}$
    if $t > 1$ **then**
      $\beta_L = \rho / \rho_L$
      $p_{I:t} = z_{I:t} + \beta_L p_{I:t}$
    else
      $p_{I:t} = z_{I:t}$
    end if
    barrier
    $q = Ap$
    barrier
    reduce pdotq $\leftarrow_{+} p_{I:t}^T q_{I:t}$
    $\alpha_L = \rho/\text{pdotq}$
    $x_{I:t} = x_{I:t} + \alpha_L p_{I:t}$
    $r_{I:t} = r_{I:t} - \alpha_L q_{I:t}$
    reduce $||r||_2^2 \leftarrow_{+} ||r_{I:t}||_2^2$
    $\text{resnorm}_L = \sqrt{||r||_2^2/\text{bnrm2}}$
    if $\text{resnorm}_L < tol$ **stop**
    $\rho_L = \rho$
  end for
end for

Conjugate gradient iteration, Xeon

Xeon, time to solve of $Ax = b$.

Test problem	Time (s) nt=1	nt=16	nt=32	Speedup, T_{nt}/T_1 nt=16	nt=32
CRS					
3D_heat_small	0.17	**0.03**	0.06	5.27	2.95
3D_heat_med	0.85	**0.20**	0.24	4.17	3.50
3D_heat_large	3.36	**0.87**	0.94	3.84	3.55
3D_elas_small	1.13	**0.25**	0.33	4.50	3.43
3D_elas_med	8.59	**2.51**	2.67	3.43	3.22
3D_elas_large	36.34	**10.39**	10.72	3.50	3.39
NUMA CRS					
3D_heat_small	0.17	0.02	**0.01**	7.75	11.59
3D_heat_med	0.87	0.20	**0.05**	4.43	16.48
3D_heat_large	3.45	0.86	**0.46**	4.00	7.48
3D_elas_small	1.15	0.20	**0.07**	5.81	16.44
3D_elas_med	8.83	2.57	**1.28**	3.44	6.90
3D_elas_large	36.49	10.76	**5.69**	3.39	6.41

Conjugate gradient iteration, Xeon Phi

Xeon Phi, time to solve of $Ax = b$.

Test problem	Time (s) nt=1	nt=60	nt=240	Speedup, T_{nt}/T_1 nt=60	nt=240
CRS					
3D_heat_small	1.80	**0.65**	1.34	2.76	1.34
3D_heat_med	7.26	**1.07**	1.50	6.81	4.85
3D_heat_large	28.29	2.66	**2.34**	10.62	12.07
3D_elas_small	10.23	**3.22**	7.46	3.18	1.37
3D_elas_med	72.53	**6.17**	11.24	11.76	6.45
3D_elas_large	303.35	**9.87**	15.85	30.73	19.13
NUMA CRS					
3D_heat_small	1.75	**0.10**	0.13	17.24	13.65
3D_heat_med	7.34	**0.24**	0.33	30.21	21.94
3D_heat_large	28.70	0.80	**0.70**	35.87	40.78
3D_elas_small	10.19	**0.35**	0.59	28.96	17.32
3D_elas_med	73.12	1.86	**1.69**	39.34	43.22
3D_elas_large	306.20	7.14	**4.51**	42.86	67.83

Computational complexity of Krylov methods I

Is conventional wisdom correct?

In a purely shared memory environment, is it correct to estimate the performance of a Krylov solver based on matrix-vector products alone?

Computational complexity of Krylov methods II

		Xeon, time (s)			Xeon Phi, time (s)		
Test problem	its	mvp	pcg	Ratio	mvp	pcg	Ratio
CRS							
3D_heat_large	270	0.73	**0.94**	77.20%	1.48	2.34	62.98%
3D_elas_large	1236	9.60	**10.72**	89.61%	8.52	15.85	53.77%
NUMA CRS							
3D_heat_large	270	0.43	**0.46**	92.30%	0.51	0.70	72.24%
3D_elas_large	1236	5.16	5.69	90.64%	3.44	**4.51**	76.16%

As the system gets larger, when properly implemented sparse matrix-vector products become more dominant on a many-core system.

Conclusions

For (large) dense systems flops have become free.

For sparse linear systems flops are still expensive, but..

- Hardware is rapidly evolving
  - gather/scatter support
  - increased memory bandwidth
  - transactional memory
  - many-core
- To take advantage of the nested parallelism (SIMD & thread & node), consider
  - data formats
  - (legacy) code
  - algorithms

Thank you for your interest!

Questions?