40 JAAR 1971 2011

Optimization on Lisa

Wim Rijks
wimr@sara.nl

Contents

- Introductory Remarks
- Support team
- Optimization strategy
- Amdahls law
- Compiler options
- An example

Optimization
Introductory Remarks

- Modern day supercomputers are still expensive; use them efficiently:
  - Design for parallelism
  - Optimize sequential execution with characteristics of modern cpus in mind.
  - Optimize for configuration of system you run on.
- Old code:
  - consider rewriting it from scratch,
  - concentrate on hotspots
  - Invest a little time cleaning up the code,
- Consider if your programming effort is worth the gain (manpower is expensive too).
- Don’t hesitate to ask assistance from SARA

Optimization
SARA Support team

Marcin Zielinski

John Donners

Walter Lioen

Jeroen Engelberts

Wim Rijks

Willem Vermin

Optimization
Profile your code

As a first step: profile your code!!

- **Very rough: $ time ./executable**
- **More refined: bracket blocks of code by timing routine: (MPI_Wtime(), date_and_time(),cpu_time())**
- **Use gprof:**
  - **Compile with flags:** -pg –g
  - **Execute your executable:** $ ./executable
  - **Generate profile:** $ gprof ./executable gmon.out

Optimization
"GPROF" profiling report

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	self calls	self s/call	total s/call	name
65.65	16.66	16.66	158	0.11	0.11	.__multigridpoissonsolver_NMOD_pois
27.90	23.74	7.08	10	0.71	1.79	.main
2.44	24.36	0.62	20	0.03	0.03	.__multigridpoissonsolver_NMOD_inipoi
1.38	24.71	0.35				._init
0.79	24.91	0.20	316932	0.00	0.00	.__multigridpoissonsolver_NMOD_inicoe
0.39	25.01	0.10	10	0.01	0.01	.rhotempmultifgm
0.39	25.11	0.10	10	0.01	0.01	.sgsvre
0.24	25.17	0.06				.cflxv
0.24	25.23	0.06				.vflxw
0.24	25.29	0.06				._start
0.16	25.33	0.04				.cflxu

Parallel optimization Strategy

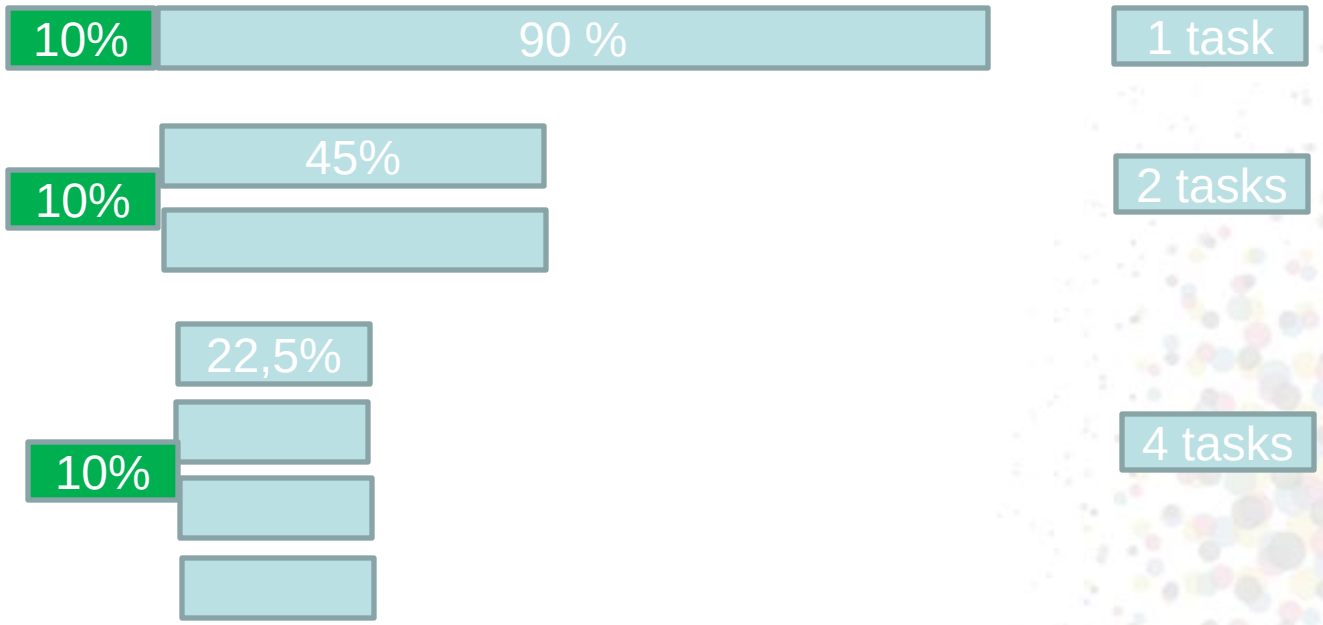
- **Choose parallelization paradigm**
  - **OpenMP:**
  - **MPI:**
  - **Hybrid (MPI + OpenMP)**
  - **Other (New developments: UPC, CAF)**
- **Keep realistic expectations**
  - **Remember Amdahl's law:**

    The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program

Parallel optimization
Amdahl's law

$$S = \frac{1}{(1-P) + P/Sp}$$

S = speedup
P = fraction parallel
Sp = speedup parallel fraction



Sequential optimization
strategy (1)

- Check literature for most efficient algorithm.
- Check if algorithm already is implemented: use existing applications or libraries!!!!
- Choose a Language
  - Don’t use interpreted code (python, perl, ….)
  - c, C++, Fortran
  - Choose a compiler, preferably vendor supplied compiler. On Lisa: INTEL compiler suite.

Sequential optimization
strategy (2)

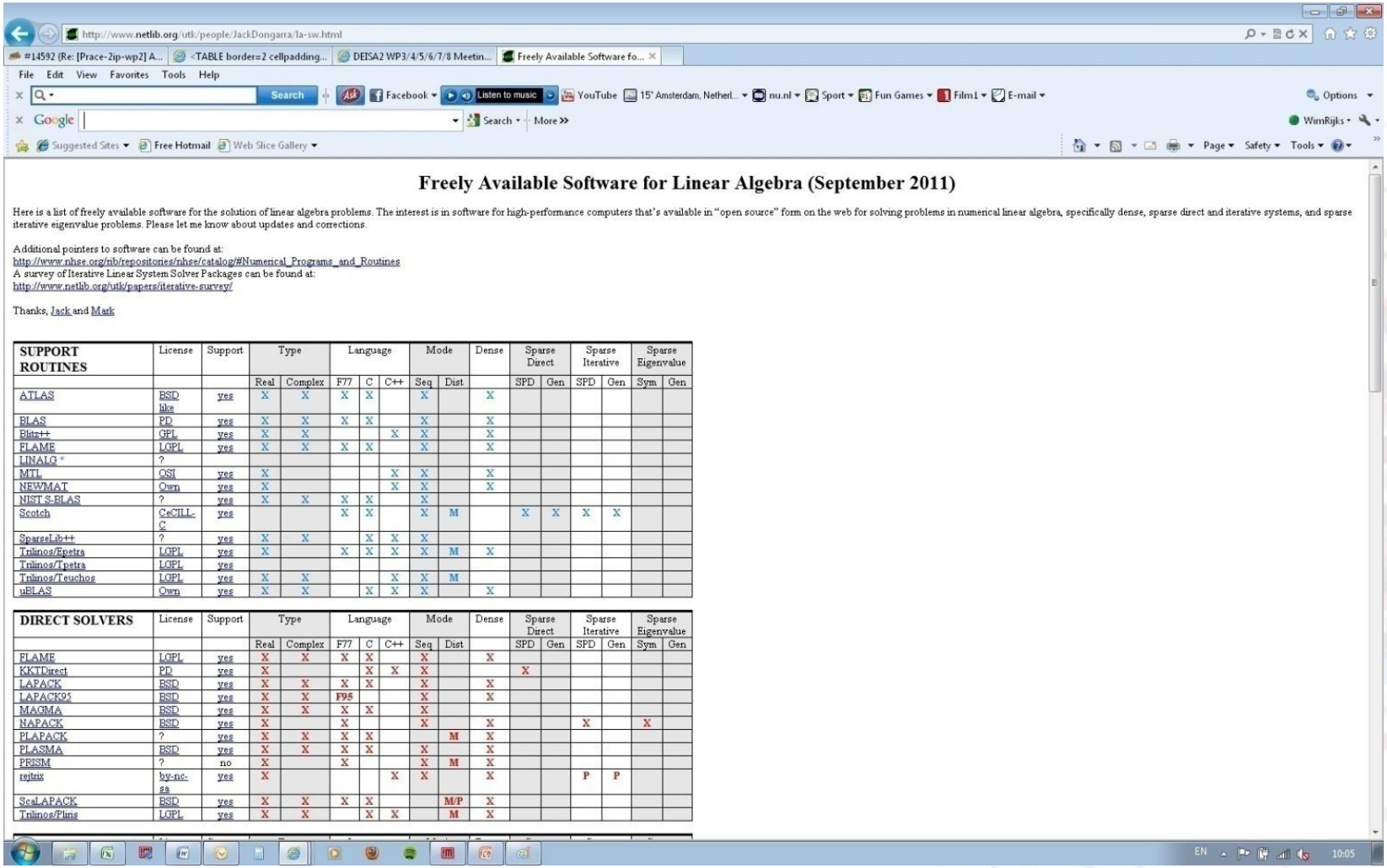
- Write clean, not to complex code:
  - Let the compiler do as much work as possible
  - But still keep in mind, while designing your code, the characteristics of modern day cpu's
- Profile your code and find the hotspots.
- Try out compiler options for optimization
  - During development and debugging: use conservative optimization flags.
  - During optimization: use more aggressive optimization. Can alter the logic of the code: keep checking consistency of results
- Reassess critical pieces of code

Sequential optimization
Use libraries

- For Blas, lapack, FFT routine use vendor libraries like MKL (Intel) or ESSL (IBM)
- Check out other numerical libraries
  - (NAG, IMSL, ScaLapack, FFTW,PETSc, MUMPS)
  - Have a look on the internet for a list of libraries and their contents http://www.netlib.org/utk/people/JackDongarra/la-sw.html (table of Support routines, Direct solvers, Sparse direct solvers, Preconditioners, Sparse iterative solvers, Sparse eigenvalue solvers)

Sequential optimization

Use libraries



http://www.netlib.org/utk/people/JackDongarra/la-sw.html

#14592 (Re: [Prace-2ip-wp2] A... | <TABLE border=2 cellpadding... | DEISA2 WP3/4/5/6/7/8 Meetin... | Freely Available Software fo...

File Edit View Favorites Tools Help

Search | Facebook | Listen to music | YouTube | 15° Amsterdam, Netherl... | nu.nl | Sport | Fun Games | Film1 | E-mail | Options

Google | Search | More | WimRijks

Suggested Sites | Free Hotmail | Web Slice Gallery | Page | Safety | Tools

Freely Available Software for Linear Algebra (September 2011)

Here is a list of freely available software for the solution of linear algebra problems. The interest is in software for high-performance computers that's available in "open source" form on the web for solving problems in numerical linear algebra, specifically dense, sparse direct and iterative systems, and sparse iterative eigenvalue problems. Please let me know about updates and corrections.

Additional pointers to software can be found at:
http://www.nhse.org/nb/repositories/nhse/catalog/#Numerical_Programs_and_Routines
A survey of Iterative Linear System Solver Packages can be found at:
http://www.netlib.org/utk/papers/iterative-survey/

Thanks, Jack and Mark

SUPPORT ROUTINES	License	Support	Type		Language			Mode		Dense	Sparse Direct		Sparse Iterative		Sparse Eigenvalue	
			Real	Complex	F77	C	C++	Seq	Dist		SPD	Gen	SPD	Gen	Sym	Gen
ATLAS	BSD like	yes	X	X	X	X		X		X						
BLAS	PD	yes	X	X	X	X		X		X						
Blitz++	GPL	yes	X	X			X	X		X						
FLAME	LGPL	yes	X	X	X	X		X		X						
LINALG °	?															
MTL	OSI	yes	X				X	X		X						
NEWMAT	Own	yes	X				X	X		X						
NIST S-BLAS	?	yes	X	X	X	X		X								
Scotch	CeCILL-C	yes			X	X		X	M		X	X	X	X		
SparseLib++	?	yes	X	X		X	X	X								
Trilinos/Epetra	LGPL	yes	X		X	X	X	X	M	X						
Trilinos/Tpetra	LGPL	yes														
Trilinos/Teuchos	LGPL	yes	X	X			X	X	M							
uBLAS	Own	yes	X	X		X	X	X		X						

DIRECT SOLVERS	License	Support	Type		Language			Mode		Dense	Sparse Direct		Sparse Iterative		Sparse Eigenvalue	
			Real	Complex	F77	C	C++	Seq	Dist		SPD	Gen	SPD	Gen	Sym	Gen
FLAME	LGPL	yes	X	X	X	X		X		X						
KKTDirect	PD	yes	X			X	X	X			X					
LAPACK	BSD	yes	X	X	X	X		X		X						
LAPACK95	BSD	yes	X	X	F95			X		X						
MAGMA	BSD	yes	X	X	X	X		X								
NAPACK	BSD	yes	X		X			X		X			X		X	
PLAPACK	?	yes	X	X	X	X			M	X						
PLASMA	BSD	yes	X	X	X	X		X		X						
PRISM	?	no	X		X			X	M	X						
rejtrix	by-nc-sa	yes	X				X	X		X			P	P		
ScaLAPACK	BSD	yes	X	X	X	X			M/P	X						
Trilinos/Pliris	LGPL	yes	X	X		X	X		M	X						

Sequential optimization
Characteristics of Modern CPUs

- Pipelined floating point processor
  - Very efficient when performing same operation on large arrays
  - After certain startup time they can produce a result each clock cycle
- Two or more levels of cache memory
  - Accessing vectors in memory with stride 1 is very important
  - Try to reuse data that is already in cache

Sequential optimization
Compiler options

- **Invocation:**
  - icc, iCC, ifort
- **Parallelization:**
  - OpenMP flags: -parallel –openmp –par-report
  - MPI: load openmpi + mpif90,mpicc,…
- **Optimization:**
  - -O[n] –ftz –ax<proc> -x<proc> –march=<proc> -mtune=<proc> -ip -ipo
  - For proc: SSE3, SSE4.2

Machine optimization

- Use processor affinity
- check if it is benificial to leave one core per node free for system tasks.
- run multiple (serial) programs on same node

Sequential optimization Example: matrix multiply

Demonstrate effect of:

- **Choice of compiler**
- **Choice of compiler options**
- **Access memory with stride > 1**
- **Using MKL Library**
- **Using fortran intrinsic function**
- **Influence of manual optimization**

Sequential optimization
Example: Matrix multiply

```
! Matrix multiply C = A * B
  c = 0.d0
  call cpu_time(t1)
  do i=1,ndim
    do j=1,ndim
      do k=1,ndim
        c(i,j) = c(i,j) + a(i,k)*b(k,j)
      enddo
    enddo
  enddo
  call cpu_time(t2)
  gflops = 2.d0*ndim*ndim*ndim/(giga*(t2-t1))
  print *,c(1,1),c(ndim,ndim)
  write(6,'("Loop order: i,j,k  CPU time: ",1pd14.7,"   GFLOPS: ",f10.3)')
      t2-t1, gflops
```

Sequential optimization

Example: loop order + compiler options

	gfortran	gfortran	Ifort (default)	Ifort (default)	Ifort (default)
	-m64 –O3 –ffree-format	-m64 -O5 -ffree-form	-O2 -free	-O3 -free	-O3 –free –axSSE4.2
k,j,i	13,95	15,87	14,87	3,64	3,64
j,k,i	12,82	14,23	12,64	3,91	3,86
i,j,k	99,75	101,91	12,62	4,10	3,95
j,i,k	100,38	101,92	12,63	3,91	3,86
i,k,j	228,25	227,23	12,64	3,91	3,85
k,i,j	228,73	228,06	12,64	3,91	3,87

Optimization strategy
Example: using libraries

	Ifort -O3 –free –axSSE4.2	Ifort -O3 –free –axSSE4.2	Gfortran –O3 -free-form –m64
	MKL (sequential)	BLAS	ATLAS
j,k,i	3,86	4,15	14,70
dgemm	2,07	20,27	2,20
matmul	3,78	4,00	13,35

Optimization strategy
Example: Matrix multiply

```
call cpu_time(t1)
do i=1,ndim
  do j=1,ndim
     a_trans(j,i) = a(i,j)
  enddo
enddo
do i=1,ndim
  do j=1,ndim
    do k=1,ndim
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
       c(i,j) = c(i,j) + a_trans(k,i)*b(k,j)
    enddo
  enddo
enddo
call cpu_time(t2)
```

Optimization

An example: matrix multiply

- Compiled with gfortran

  gfortran –O3 –m64 –free-form

- Original code:100,4 secs
- With a_trans :15,07 secs

Questions