Some Notes on BLAS, SIMD, MIC and GPGPU

(for Octave and Matlab)

Christian Himpe (christian.himpe@uni-muenster.de)

WWU Münster
Institute for Computational and Applied Mathematics

Software-Tools für die Numerische Mathematik
15.10.2014

About[1]

- BLAS & LAPACK
- Affinity
- SIMD
- Automatic Offloading

[1] Get your Buzzword-Bingo ready!

BLAS & LAPACK

BLAS (Basic Linear Algebra System)

- Level 1: vector-vector operations
  (dot-product, vector norms, generalized vector addition)
- Level 2: matrix-vector operations
  (generalized matrix-vector multiplication)
- Level 3: matrix-matrix operations
  (generalized matrix-matrix multiplication)

LAPACK (Linear Algebra Package)

- Matrix Factorizations: LU, QR, Cholesky, Schur
- Least-Squares: LLS, LSE, GLM
- Eigenproblems: SEP, NEP, SVD

- Default (un-optimized) netlib reference implementation.
- Used by Octave, Matlab, SciPy/NumPy (Python), Julia, R.

MKL

Intel's implementation of BLAS and LAPACK.

MKL (Math Kernel Library)

- can offload computations to XeonPhi
- can use OpenMP
- provides additionally FFT, libm and 1D-interpolation

- Current Version: 11.0.5 (automatic offloading to Phis)
- Costs (we have it, and Matlab ships with it, too)
- Go to: `https://software.intel.com/en-us/intel-mkl`

ACML

AMD doesn't want to be left out.

ACML (AMD Core Math Libraries)

- Can use OpenCL for automatic offloading to GPUs
- Special version to exploit FMA(4) instructions
- Choice of compiled binaries (GFortran, Intel Fortran, Open64, PGI)

- Current Version: 6 (automatic offloading to GPUs)
- Free (but not open source)
- Go to: `http://developer.amd.com/tools-and-sdks/cpu-development/amd-core-math-library-acml`

OpenBLAS

Open Source, the third kind.

OpenBLAS

- Fork of GotoBLAS
- Good performance for dense operations (close to the MKL)
- Can use OpenMP and is compiled for specific architecture

- Current Version: 0.2.11
- Open source (!)
- Go to: `http://github.com/xianyi/OpenBLAS`

FlexiBLAS

So many BLAS implementations, so little time...

FlexiBLAS to the rescue

- Abstraction Layer for BLAS
- Compile your program with FlexiBLAS
- And decide at runtime via environment variable which to use

- Developed at MPI Magdeburg
- By M.Köhler and J.Saak
- Go to:
  `http://www.mpi-magdeburg.mpg.de/projects/flexiblas`

Matlab

Runtime switching of LAPACK and BLAS backends:

Before starting Matlab, run

```
export BLAS_VERSION="/path/to/my/blas.so"
export LAPACK_VERSION="/path/to/my/lapack.so"
```

After starting Matlab, check

```
version -'blas'
version -'lapack'
```

Octave

Switching of LAPACK and BLAS backends:

1. Use `export LD_PRELOAD=‘‘/path/to/blas.so’’`
   Instructions can be found at:
   `http://devblogs.nvidia.com/parallelforall/drop-in-acceleration-gnu-octave/`
2. Build Octave with FlexiBLAS
   Instructions can be found at:
   `http://www.flaterco.com/kb/Octave.html`
3. Use `update-alternatives` (in Ubuntu)
   Instructions can be found at:
   `http://luiseth.wordpress.com/2012/04/08/accelerate-your-matrix-computations-with-acml-on-kubuntu-11-10/`
   (This will likely not work with MKL)

Affinity (Bouncing Cow Syndrome)

Kernel Scheduler reassignes tasks / threads to varying cores.
Why? Design-Principles, Multitasking, Energy-Saving, ...
Problem: Cache reuse and locality.

Vocabulary:

- SMT, Simultaneous MultiThreading
- HT, HyperThreading (Intel's SMT)
- CMT, Clustered MultiThreading (AMD's not-really-SMT)

- One physical Core = 2 Logical Cores
- HT will rarely help you compute faster[2]

[2] Except on In-Order CPUs like Intel's Atoms N270, N570 etc

Stay!

Set CPU affinity (Linux): `taskset -c 0,2,4,6 ./myprog`

Up to 5% more performance[3] due to pinning;
and a more tidy `htop` ☺

Alternatively, OpenMP 4.0 has: `OMP_PROC_BIND=true`

[3] Ultimately, gains depend on the scheduler

Previously, on SIMD

SIMD (Single Instruction Multiple Data)

x86[4] SIMD history:

1. MMX (8x 64-bit Regs, Integer Only)
2. 3DNow (AMD's deadend)
3. SSE (8x 128-bit Regs, Integer & Floats)
4. SSE2 (MMX $\rightarrow$ SSE)
5. SSE3 (Horizontal)
6. SSSE3 (DP Addons)
7. SSE4.1/.2/a (More Tools)
8. AVX (16x 256-bit Regs, Integer & Floats)
9. FMA3 / FMA4 (Fused-Multiply-And-Add)
10. AVX2 (includes FMA3, 256-bit Types)

[4] ARM also has SIMD extensions called NEON

AVX

Some Notes on AVX (Advanced Vector eXtensions)

- does not sport `vdppd`.
- is basically 2 SSE units,
- thus there are no "horizontal" operations over all packed.
- ensures all previous SSE extensions.
- AVX2 does not fix this.

Check it yourself:

```
typedef double vector __attribute__ ((vector_size(32)));
double dot(const vector& a,const vector& b)
{
 const vector c = a*b;
 return (c[0]+c[1])+(c[2]+c[3]);
}
```

at `http://gcc.godbolt.org` with gcc 4.9 -Os -mavx

FMA

Some Notes on FMA (Fused Multiply and Add)

- This is a hardware FLOP
- FMA4 (AMD) $r = a * b + c$
- FMA3 (Intel & AMD) $r = a * b + c$, $r = a \vee b \vee c$
- FMA3 is part of AVX2
- C99 has an `fma` library function (in `math.h`)

Check it yourself:

```
typedef double vector __attribute__ ((vector_size(32)));
vector dot(const vector& a,const vector& b,const vector& c)
{
 return a*b + c;
}
```

at `http://gcc.godbolt.org` with gcc 4.9 -Os -mfma / -mfma4

Xeon Phi

Intel's answer to GPGPU computing.

Xeon Phi is the brand for what Intel calls MIC (Many Integrated Core).

- Basically a computer in a computer.
- up to 8 inside a single node.
- 57-61 cores, 6-16GB memory models
- Each core is of Atom (N570?) architecture with 4 hardware threads.
- AVX-512 (this may be just 4 SSE units) aka IMCI (not fully AVX compatible)

Automatic Offload (MKL $\rightarrow$ PHI)

Without explicit hints, pragmas etc, the MKL can offload computations to the MIC(s) automatically.

Setting the environment variable:

```
MKL_MIC_ENABLE=1
```

enables **Automatic Offloading**.

Controlling MKL AO

MKL Automatic offloading can be controlled using environment variables.

(MKL's) OpenMP core usage is controlled by:

```
OMP_NUM_THREADS=8
```

similarly the MIC core usage is controlled:

```
MIC_OMP_NUM_THREADS=236
```

More controls for offloading are documented here:
`https://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mkl_userguide_lnx/GUID-3DC4FC7D-A1E4-423D-9C0C-06AB265FFA86.htm`

HSA[5] + hUMA

- AMD's (current) next big thing.
- Minimum is GCN 1.0 / 1.1 (Kabini / Kaveri APUs)
- HSA = Heterogeneous System Architecture (aka CPU + GPU)
- hUMA = hetergoenous Unified Memory Access (think truly Shared Memory)
- Here is the Kool-Aid: `http://www.slideshare.net/AMD/amd-heterogeneous-uniform-memory-access`

- The Good: Less Communication, and easier parallelization.
- The Bad: Current Hardware pretty restrictive.
- The Ugly: Sparsely documented.

[5] http://www.hsafoundation.com

Automatic Offload (ACML → GPU)

Requires `OpenCL`:

Choose:

- `libcl` (Arch name) by nVidia; supports OpenCL 1.0
- `libopencl` (Arch name) by AMD; supports OpenCL 1.1
- Intel has one, too

The ACML(6) then automatically offloads to the GPU;
uses internally clBLAS (`http://github.com/clMathLibraries/clBLAS`).

Controlling ACML AO

ACML Automatic Offloading can be controlled via ACMLscript

ACMLscript = LUA

Choose Resources and Heuristics

More info on ACMLscript:
`http://developer.amd.com/community/blog/2014/05/06/acmlscript/`

tl;dl

All in all:

- Know your BLASs to get more performance
- Pin tasks to avoid bouncing
- Let the compiler do SIMD
- Offloading enables parallelization without specialized code

There is more:

- ATLAS (Automatically Tuned Linear Algebra Software)
- nvBLAS (CUDA)
- ViennaCL
- OpenACC (2.0)

That's all folks