Anatomy of a Globally Recursive Embedded LINPACK Benchmark

Jack Dongarra and Piotr Luszczek

Batteries included. Some assembly required.
ARM Landscape

- **Architecture**
  - ARM11, Cortex A8, A9, A15
- **ISA**
  - ARMv6, ARMv7
- **Floating-point**
  - VFP11, VFPv3, VFPv4
  - Pipelined and not pipelined
- **Profiles**
  - Application, Real-time, Microcontroller
- **Implementations (with options and secrete sauce)**
  - Qualcomm Snapdragon
    - Scorpion, Krait
  - OMAP
  - Samsung Exynos 3, 4, 5
Why?
Famous MATLAB's Easter Egg

- MATLAB>why
  Jack made me do it.
I Thought iOS 4 had Accelerate Framework

- With iOS 4 Apple brought Accelerate Framework
- Mac OS X includes optimized ATLAS inside Accelerate
- ATLAS port to ARM is fast enough on Linux
- But in reality...
BLAS on iPad?

- Accelerate Framework on iOS 4 did not work
- No ATLAS for iDevices
  - Cross compilation
  - No explicit compiler invocation
  - App signing requirements
What is the iPad Hardware?

- We know something about ISA
  - Must be disclosed for low-level programming
- Hardware features are only approximate
  - May give advantage in a crowded market place
- The most important spec is what the software can use
  - Actual vs. effective hardware parameters
- Let's use micro- and nano-benchmarking for discovery
  - Using technique called pointer chasing (or chaining)
Discovery of the Size of TLB Level 1

![Graph showing the relationship between working set size and TLB access time.](image-url)
Discovery of TLB Page Size

![Graph showing access latency per each access (micro-seconds) versus working set size for different architectures: ARM OMAP3 ARMv5, Intel Itanium 2, and Intel Nehalem EP. The graph illustrates the latency increase as the working set size grows.]
Performance Prerequisite: L1 Cache DGEMM

- Model-based optimization
  - Schur complement update is
    \[ C \leftarrow C - A*B \]
  - A is m by k, B is k by m, C is m by n
    - All must fit in register file
    - All must fit in L1 Cache
  - Total flops is 2*m*n*k
  - Total operations: \(2*m*n*k + (m*n + m*k + n*k)\)
- Maximize: \((\text{total flops}) / (\text{total operations})\)
  - There is an analytical solution but enumeration works too
  - Optimal register allocation: 3, 7, 1
  - Optimal cache allocation: 40, 40, 4
C Compiler vs. Optimized Code

- Compiler doesn't know optimal blocking, register allocation
  - Even if you do
  - Even if you try to tell it
- Compilers don't have enough information from the programmer
  - Language semantics are to blame
  - Compiler specific extensions are a must
    - __align__, __restrict__, -fast, -f_forget_aboutaccuracy
- Compilers don't trust the programmer
- Compilers think they're smarter than the programmer
- In the end
  - Compilers are here so we don't have to use assembly
Let's Do Assembly ... in Python

- Simple adder
  Func = Function("add2numbers", ("a", Ptr_Int32), ("b", Ptr_Int32))
  a = Pointer(LoadArg(1))
  b = Pointer(LoadArg(2))
  AddToReg(a, b)
  Func.save("add2numbers.c")

- DGEMM fragment
  MultiLoad([a[0], a[1], a[2], a[3]], A)
  A += NB # increment pointer A
  FuseMulSub(c[0][0], a[0], b[0])

- Double buffering
  cReg1 = RegAlloc(Float64)
  cReg2 = RegAlloc(Float64)
  cReg = RegAlloc(Float64)
  i=0
  while i < 2:
    i += 1
    FuseMulAdd(cReg1, cReg2)
    Load(cReg, Ptr)
    if 1 == i: # swap buffers/regs
      cReg1 = RegAlloc(Float64)
      cReg2 = RegAlloc(Float64)
      cReg = RegAlloc(Float64)
      # RegFree() to release
Performance Bounds for DGEMM

The graph shows the performance bounds for different DGEMM implementations as a function of matrix size. The x-axis represents the matrix size, while the y-axis shows the Mflop/s. Different markers and line styles represent various implementations:

- Registers + 1 Load
- Registers + 2 Loads
- Single tile
- KBLAS
- Non Contiguous
- VLDMIA

The graph illustrates how these implementations perform under varying matrix sizes, highlighting the trade-offs and performance characteristics in DGEMM computations.
Algorithms Matter: The Need for Scaling Down

- Most algorithms use blocking
  - Block size is 32+
  - Limits small matrix performance
    - The curse of cleanup code
    - But non-blocking codes are way worse
- Recursive LU
  - Inherently sequential
  - Parallelism only in BLAS
  - Almost unbeatable on single-cores

- Two wrongs do make it right
  - Use non-blocking recursive code

- Algorithm sketch
  - Factor left part
  - Update
  - Factor right part
  - Pivot left part

- Make both cores follow the recursive code
- Partition work based on fine-grained data assignment
- Use cache coherency as a messaging layer
Putting It All Together
## Performance and Power Across Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Model</th>
<th>Performance</th>
<th>TDP</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD FireStream</td>
<td>9370</td>
<td>528</td>
<td>225</td>
<td>2.35</td>
</tr>
<tr>
<td>NVIDIA Fermi</td>
<td>M2050</td>
<td>515</td>
<td>225</td>
<td>2.29</td>
</tr>
<tr>
<td>AMD Magny-Cours</td>
<td>6180SE</td>
<td>120</td>
<td>115</td>
<td>1.04</td>
</tr>
<tr>
<td>Intel Westmere</td>
<td>E7-8870</td>
<td>96</td>
<td>130</td>
<td>0.74</td>
</tr>
<tr>
<td>Intel Atom</td>
<td>N570</td>
<td>6.7</td>
<td>8.5</td>
<td>0.79</td>
</tr>
<tr>
<td>ARM</td>
<td>Cortex-A9</td>
<td>2</td>
<td>0.5</td>
<td>4</td>
</tr>
</tbody>
</table>
Future Directions

- Sub-$100 dev-boards
  - BeagleBoard, PandaBoard, ODROID-X, ...
- Play with ARM Cortex A-15
  - Samsung Exynos 5 Dual, Quad, ...
  - Qualcomm Snapdragon S4 (Krait CPU is “like” Cortex A-15)
- Play with ARMv8 and 64-bit goodness
  - 2014?
- More fun with double-buffering
- Use vectorization-friendly storage
  - See ATLAS’ Git tree for details (access-major storage)
- Look into and/or beyond OpenGL ES
  - Shader language
  - CUDA (embedded? full?)
<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Processor</th>
<th>Implementation</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeagleBoard</td>
<td>$200</td>
<td>Cortex A9</td>
<td></td>
<td>Linux Debian, ...</td>
</tr>
<tr>
<td>PandBoard</td>
<td>$200</td>
<td>Cortex A9</td>
<td></td>
<td>Linux Debian, ...</td>
</tr>
<tr>
<td>Raspberry Pi</td>
<td>$25</td>
<td>ARM11</td>
<td></td>
<td>Linux</td>
</tr>
<tr>
<td>Cotton Candy</td>
<td></td>
<td>Cortex A9</td>
<td>Samsung</td>
<td></td>
</tr>
<tr>
<td>Mele A1000</td>
<td></td>
<td>Cortex A8 (Allwinner A10)</td>
<td></td>
<td>Android 2.3</td>
</tr>
<tr>
<td>MK802</td>
<td></td>
<td>Cortex A8 (Allwinner A10)</td>
<td></td>
<td>Android 4.0</td>
</tr>
<tr>
<td>Oval Elephant</td>
<td></td>
<td>Cortex A8 (Allwinner A10)</td>
<td></td>
<td>Android 4.0, Linaro</td>
</tr>
<tr>
<td>Mini X</td>
<td></td>
<td>Cortex A8 (Allwinner A10)</td>
<td></td>
<td>Android 2.3</td>
</tr>
<tr>
<td>Cubieboard A10</td>
<td></td>
<td>Cortex A8</td>
<td>Not pipelined</td>
<td></td>
</tr>
<tr>
<td>UG802</td>
<td></td>
<td>Cortex A9 dual-core</td>
<td></td>
<td>Android 4.0 ICS</td>
</tr>
<tr>
<td>ODROID-X</td>
<td>$129</td>
<td>Cortex A9 (Exynos 4)</td>
<td>Samsung</td>
<td>Android</td>
</tr>
</tbody>
</table>