Anatomy of a Globally Recursive Embedded LINPACK Benchmark

Jack Dongarra and Piotr Luszczek

Batteries included.
Some assembly required.

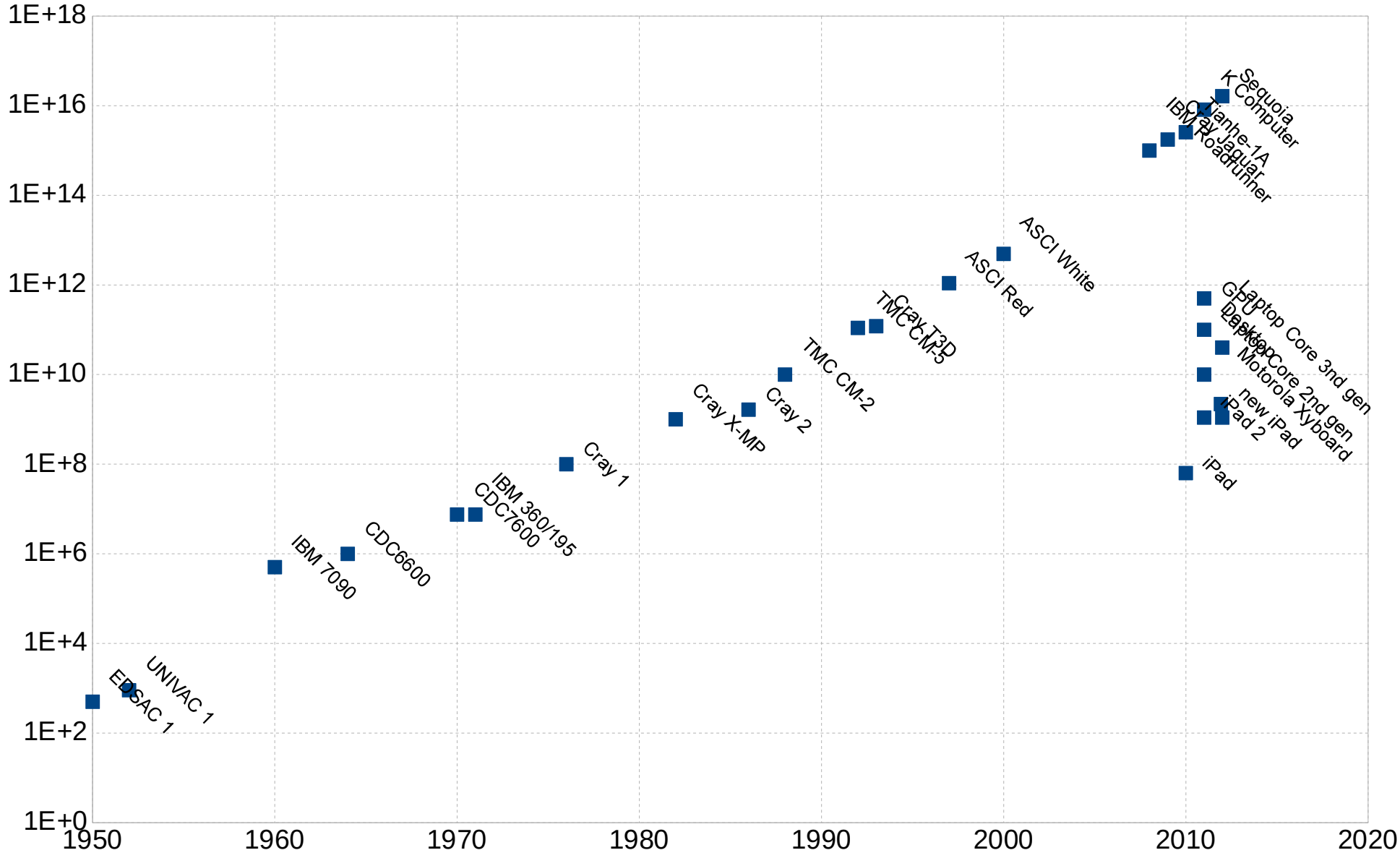
ARM Landscape

- 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.`


1E+18
1E+16
1E+14
1E+12
1E+10
1E+8
1E+6
1E+4
1E+2
1E+0
1950
1960
1970
1980
1990
2000
2010
2020
UNIVAC 1
EDSAC 1
IBM 7090
CDC6600
IBM 360/195
CDC7600
Cray 1
Cray X-MP
Cray 2
TMC CM-2
Cray T3D
TMC CM-5
ASCI Red
ASCI White
K Computer
Sequoia
Cray
Tianhe-1A
IBM
Jaguar
Roadrunner
GPU
Laptop
Laptop
Core 2nd gen
Motorola Xyboard
new iPad
iPad 2
iPad
Laptop Core 3nd gen

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…

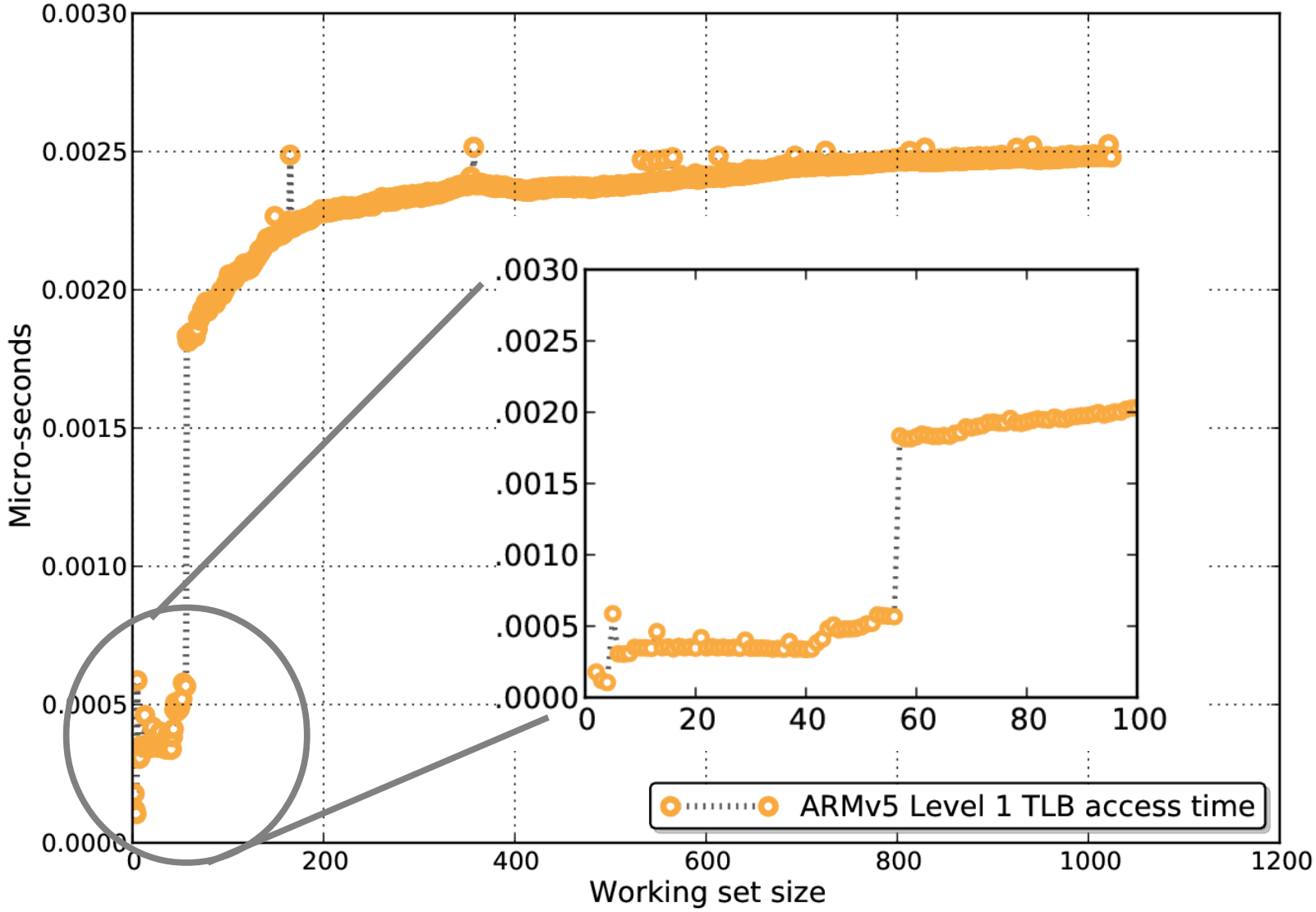
BLAS on iPad?

- Accelerate Framework on iOS 4 did not work
- No ATLAS for iDevices
  - Cross compilation
  - No explicit compiler invocation
  - App signing requirements

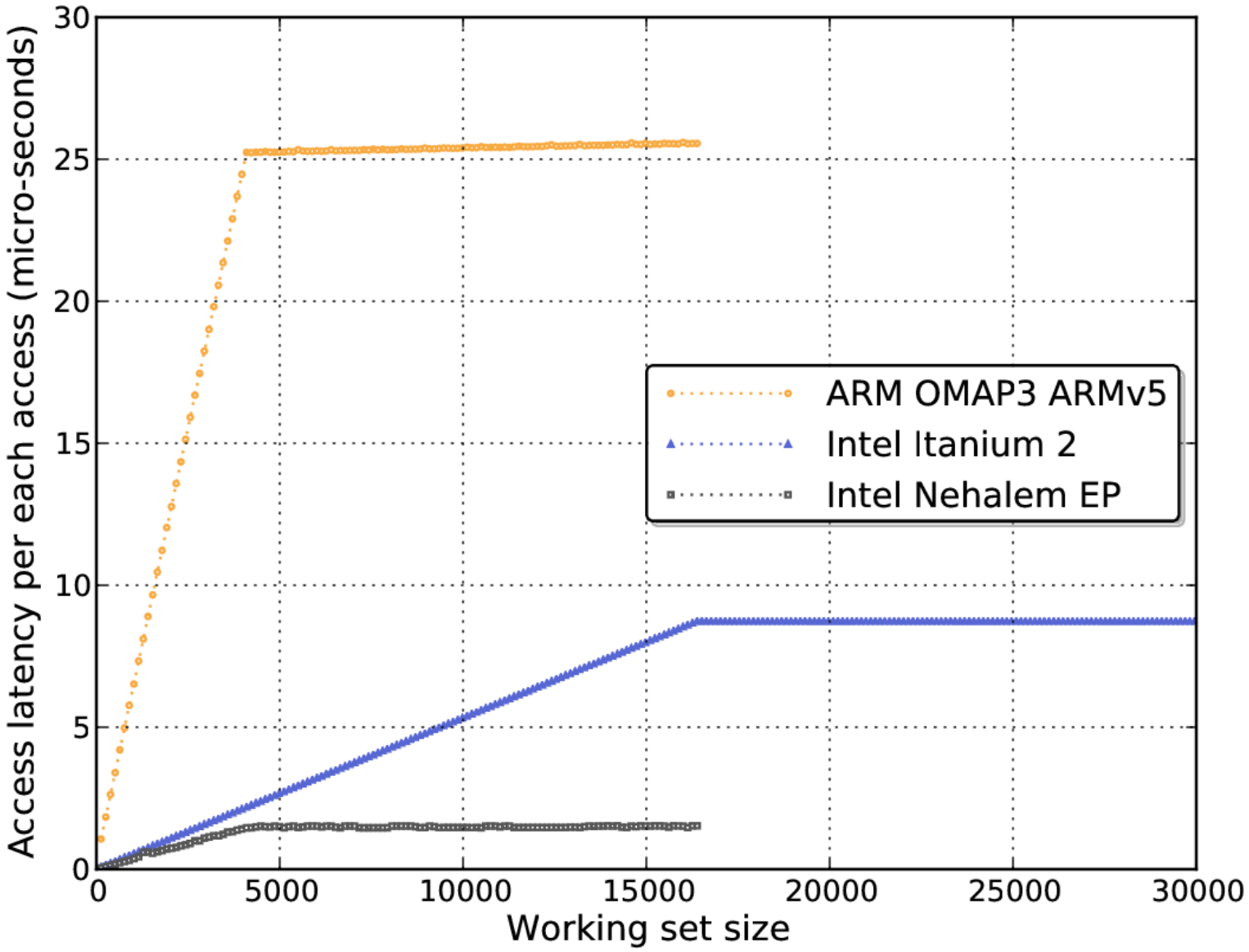
What is the iPad Hardware?

- 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



Discovery of TLB Page Size



Performance Prerequisite: L1 Cache DGEMM

- Model-based optimization
  - Schur complement update is C ← 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: (total flops) / (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_about_accuracy
- 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

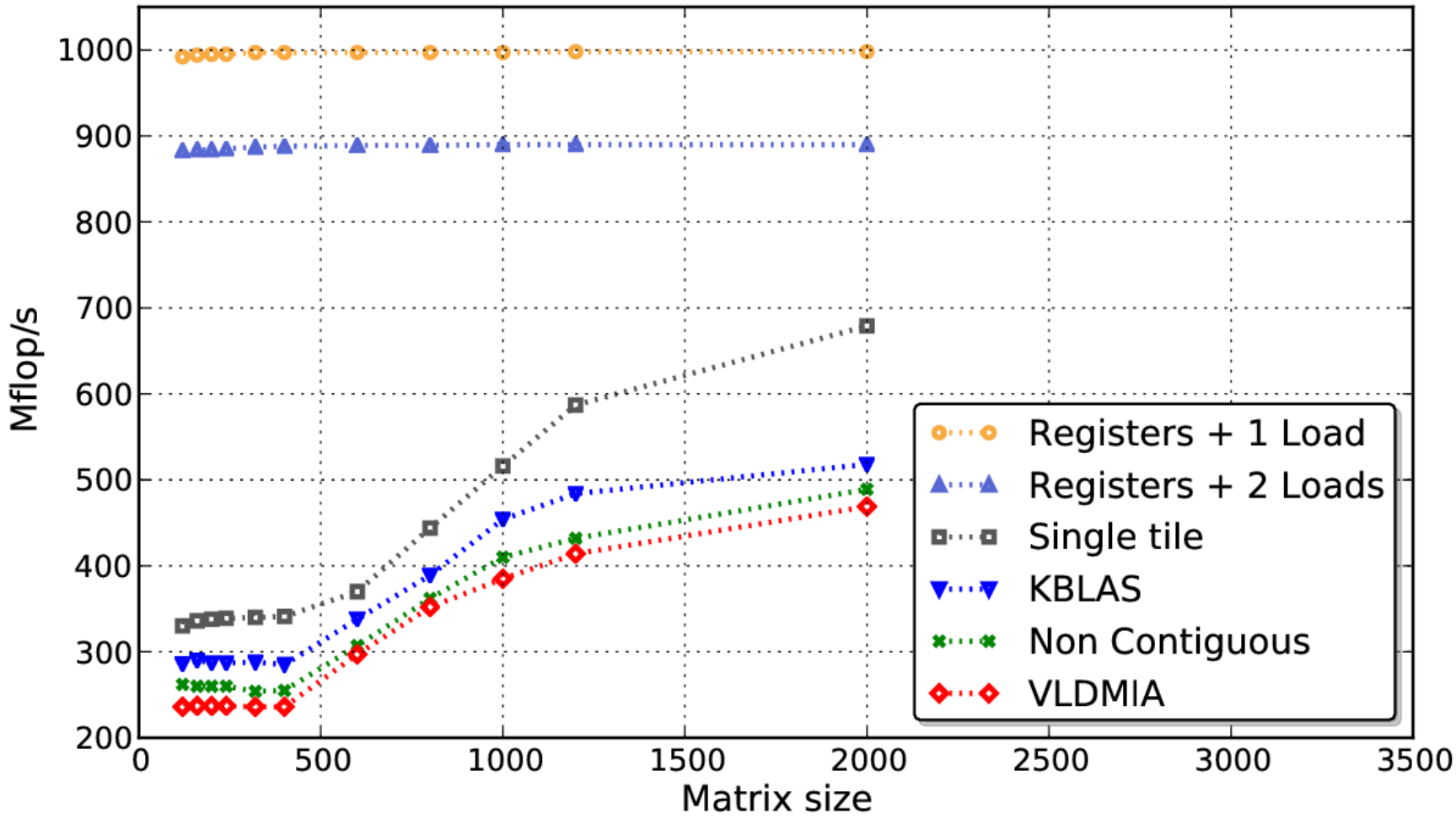
```
MultiLoad([a[0], a[1], a[2], a[3]], A)
A += NB # increment pointer A
FuseMulSub(c[0][0], a[0], b[0])
```

- 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

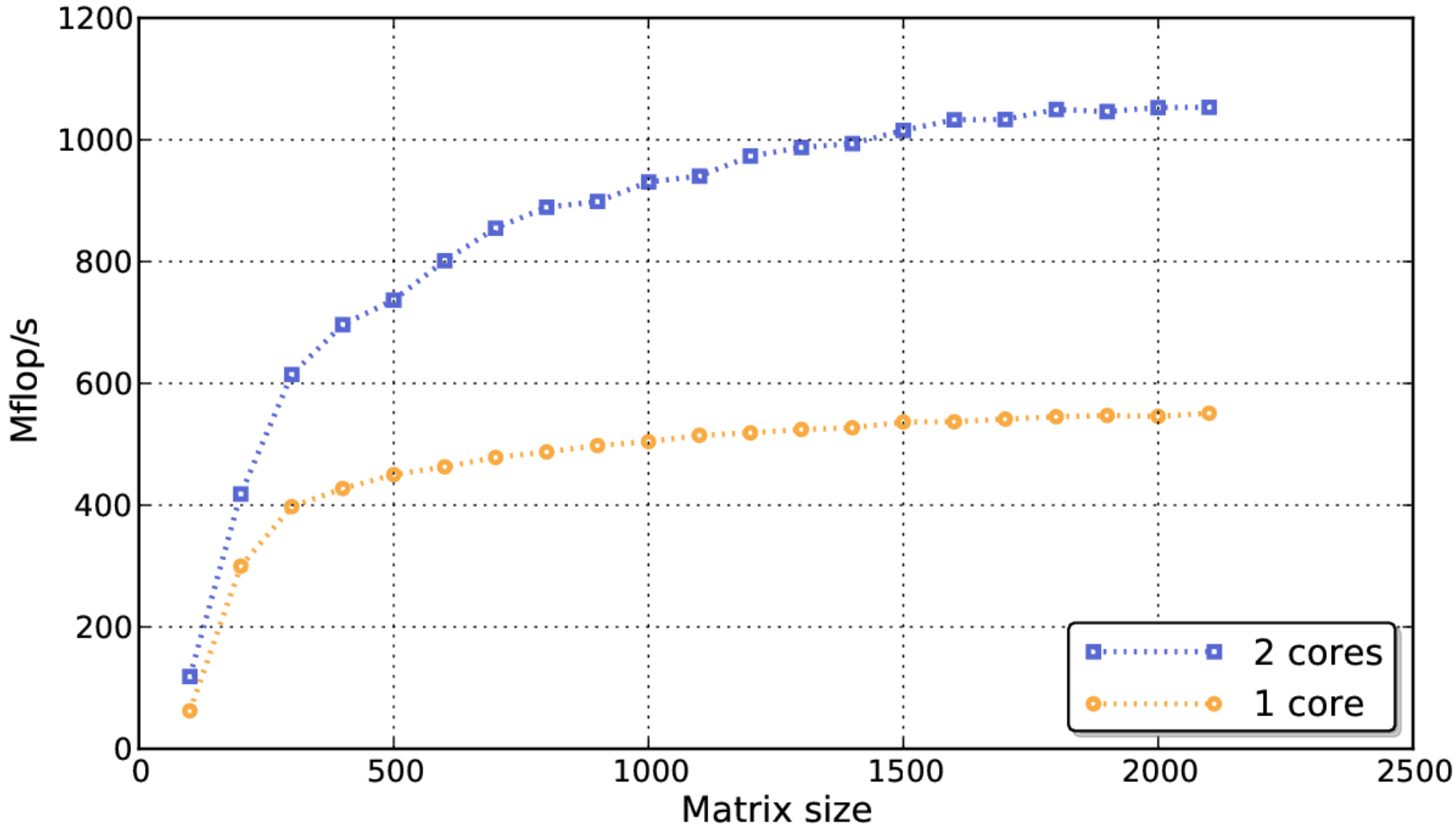


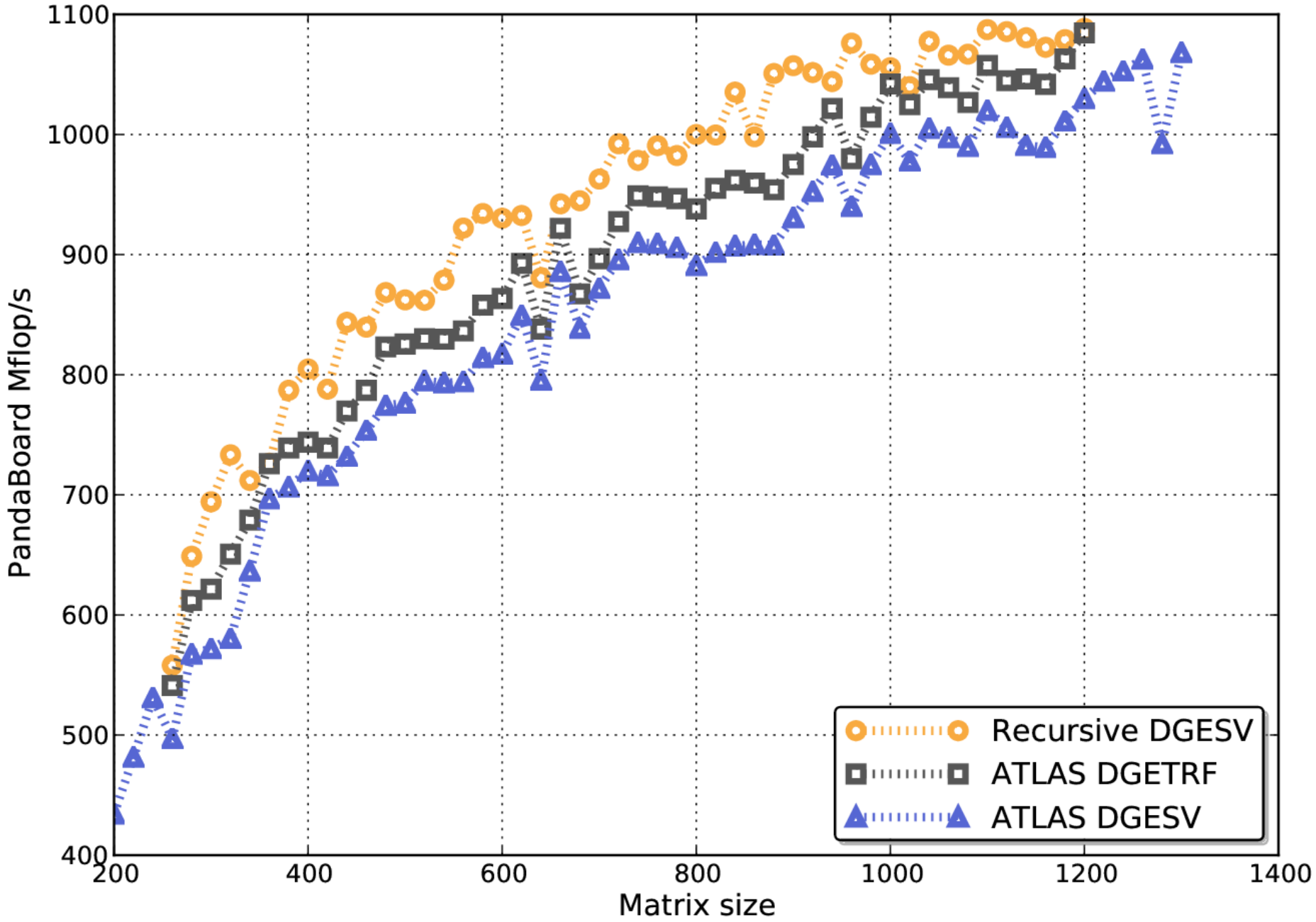
Algorithms Matter: The Need for Scaling Down

- 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




PandaBoard Mflop/s
Matrix size
1100
1000
900
800
700
600
500
400
200
400
600
800
1000
1200
1400
Recursive DGESV
ATLAS DGETRF
ATLAS DGESV

Performance and Power Across Devices

Device	Model	Performance	TDP	Efficiency
AMD FireStream	9370	528	225	2.35
NVIDIA Fermi	M2050	515	225	2.29
AMD Magny-Cours	6180SE	120	115	1.04
Intel Westmere	E7-8870	96	130	0.74
Intel Atom	N570	6.7	8.5	0.79
ARM	Cortex-A9	2	0.5	4

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?)

Sub-$100 Device Zoo

Name	Price	Processor	Implementation	OS
BeagleBoard	$200	Cortex A9		Linux Debian, ...
PandBoard	$200	Cortex A9		Linux Debian, ...
RaspberryPi	$25	ARM11		Linux
Cotton Candy		Cortex A9	Samsung	
Mele A1000		Cortex A8 (Allwinner A10)		Android 2.3
MK802		Cortex A8 (Allwinner A10)		Android 4.0
Oval Elephant		Cortex A8 (Allwinner A10)		Android 4.0, Linaro
Mini X		Cortex A8 (Allwinner A10)		Android 2.3
Cubieboard A10		Cortex A8	Not pipelined	
UG802		Cortex A9 dual-core		Android 4.0 ICS
ODROID-X	$129	Cortex A9 (Exynos 4)	Samsung	Android