1st BLIS Retreat. Austin (Texas)

Porting BLIS to new architectures
Early experiences

Francisco D. Igual

Universidad Complutense de Madrid (Spain)

September 5, 2013

BLIS design principles

BLIS = Programmability + Performance + Portability

Share experiences about:

- **Porting BLIS to different architectures:**
  - Low-power: ARM Cortex A9
  - General-purpose: Intel Sandy Bridge
  - Specific-purpose: TI C6678 DSP
- **Early experiences, results and conclusions**
- **Sequential and parallel results**
- **Some plans on extending BLIS to DMA-enabled architectures**
- **Future porting plans and architectures**

BLIS design principles

BLIS = Programmability + Performance + Portability

Disclaimer

- Based on early versions of BLIS
- Not an expert!!
  - In the target architectures
  - In BLIS
- Report early experiences. Performance can be (**will be**) improved
- The talk will not cover low level details such as micro-kernel implementations

Outline

1. ARM Cortex A9
2. Intel Sandy Bridge
3. Texas Instruments C6678 DSP
4. Integration of DMA in BLIS
    - Mapping BLIS/GotoBLAS to the C66x DSP core
    - Memory requirements. DMA
5. Conclusions

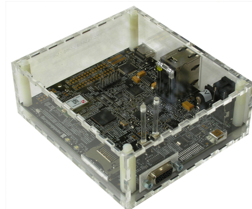
Outline

1. ARM Cortex A9
2. Intel Sandy Bridge
3. Texas Instruments C6678 DSP
4. Integration of DMA in BLIS
   - Mapping BLIS/GotoBLAS to the C66x DSP core
   - Memory requirements. DMA
5. Conclusions

Porting BLIS to ARM Cortex A9

Environment

- Tested on a PandaBoard - ARM Cortex A9
- Dual-core @ 1 Ghz, 1Gb DDR2 RAM
- OMAP4430 - 32 Kb L1, 512 Kb L2
- Ubuntu 12.04, GNU Toolchain version 4.6
- Only tested double precision, as proof of concept
    - No NEON capabilities for SIMD



- Only tuned implementation: ATLAS (`http://www.vesperix.com/arm/atlas-arm/`)
    - Compilation time: 1 day
    - No cross-compilation possible

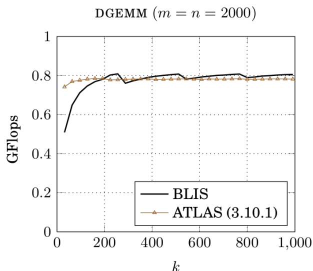
Porting BLIS to ARM Cortex A9. Hands-on

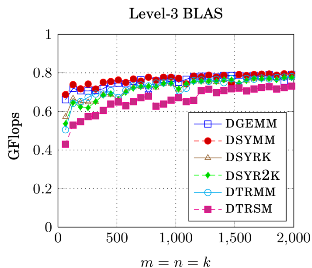
Configuring BLIS for ARM

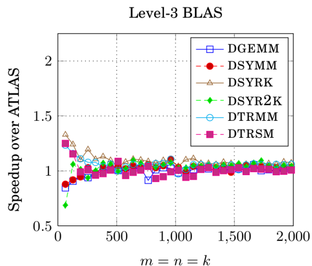
1. Create a new configuration folder (`config/pandaboard`)
2. Tune block size parameters and compiler flags (`-O3 -march=armv7-a -mtune=cortex-a9 -mfpu=neon -mfloat-abi=hard`)
3. Configure, compile and install (`./configure pandaboard && make && make install`)
4. Developed three micro-kernels: naive, assembly and plain C
    - In the end, plain C with basic optimizations won

- Compilation time: around 6 minutes
- Cross-compilation possible

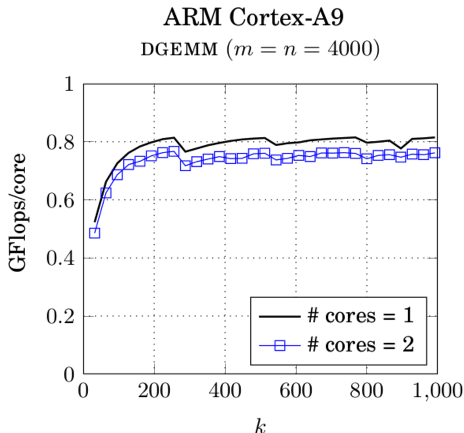
Porting BLIS to ARM Cortex A9. Sequential results







Porting BLIS to ARM Cortex A9. Parallel results



Lessons learned

After porting to ARM

- BLIS can be ported to architectures other than x86. BLIS seems portable
- Beat ATLAS for ARM using a microkernel written in C. BLIS is fast
- Fast and cross compilation

After porting to Intel Sandy Bridge

- ...
- ...

After porting to TI C6678

- ...
- ...

Outline

1. ARM Cortex A9
2. Intel Sandy Bridge
3. Texas Instruments C6678 DSP
4. Integration of DMA in BLIS
   - Mapping BLIS/GotoBLAS to the C66x DSP core
   - Memory requirements. DMA
5. Conclusions

Porting BLIS to Intel Sandy/Ivy Bridge

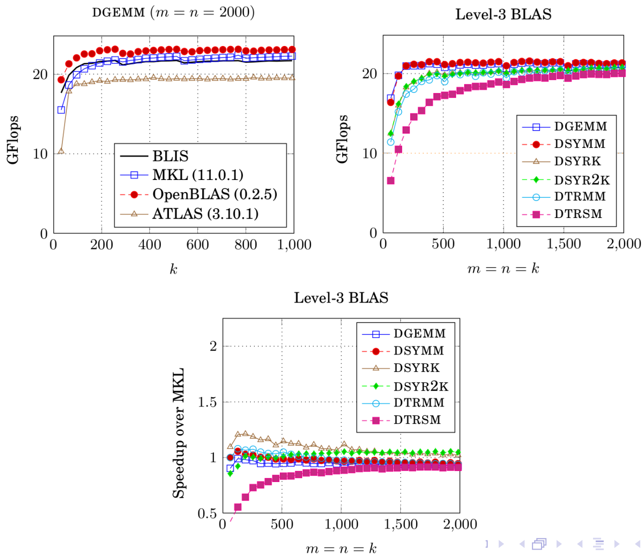
Environment

- Quad-core Intel E3-1220 @ 3.1 Ghz, 8 Gb DDR3 RAM
- Ubuntu 12.04, GNU Toolchain 4.6
- Only tested double precision, as proof of concept
- Compared with OpenBLAS, MKL and ATLAS

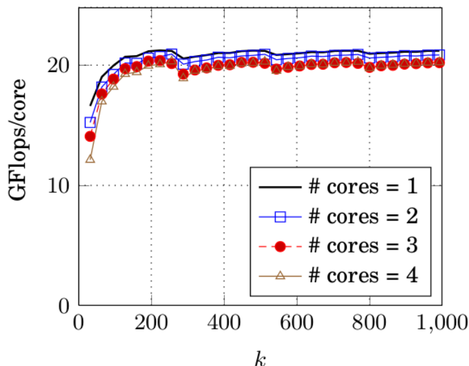
Configuring BLIS for Intel SBR

1. Create a new configuration folder (`config/sandybridge`)
2. Tune block size parameters and compiler flags (`-O3 -mavx -march=nocona -mfpmath=sse`)
3. Configure, compile and install (`./configure sandybridge && make && make install`)
4. Developed micro-kernel: inline assembly with **AVX intrinsics**

Porting BLIS to Intel Sandy/Ivy Bridge. Sequential results



Porting BLIS to Intel Sandy/Ivy Bridge. Parallel results



Lessons learned

After porting to ARM

- BLIS can be ported to architectures other than x86. BLIS seems portable
- Beat ATLAS for ARM using a microkernel written in C. BLIS is fast
- Fast and cross compilation

After porting to Intel Sandy Bridge

- Beat ATLAS by a wide margin. BLIS is fast
- Beated by OpenBLAS and MKL. BLIS can be faster

After porting to TI C6678

- ...
- ...

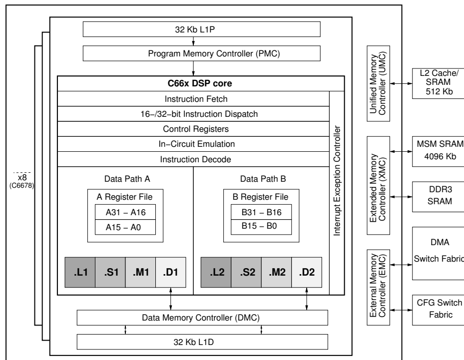
Outline

1. ARM Cortex A9
2. Intel Sandy Bridge
3. Texas Instruments C6678 DSP
4. Integration of DMA in BLIS
   - Mapping BLIS/GotoBLAS to the C66x DSP core
   - Memory requirements. DMA
5. Conclusions

TI C6678 architecture. C66x core

TI C6678 highlights

- Eight C66x cores, 1 Ghz, 10W power dissipation

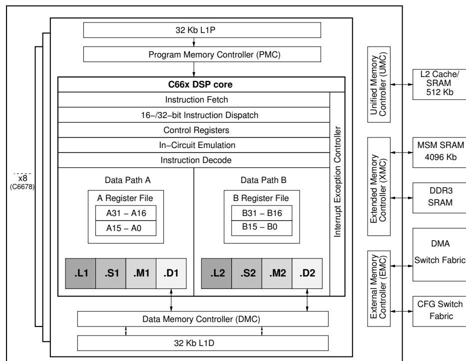


- **8-way VLIW architecture**
- **8 functional units in two sides**:
  - **M**: multiplication
  - **D**: load/store
  - **L and S**: addition/branch
- **SIMD up to 128-bit vectors**:
  - **M**: 4 SP multiplies/cycle
  - **L and S**: 2 SP add/cycle
- **8 MAC per cycle**:
  - 128 GFLOPs @ 1Ghz

TI C6678 architecture. C66x core

TI C6678 highlights

- Eight C66x cores, 1 Ghz, 10W power dissipation



- **On-chip memory**
  - **L1D + L1P**: 32 Kb
  - **L2**: 512 Kb
  - **MSMC**: 4096 Kb (shared by cores)
  - **Configurable as RAM or cache**
- **Off-chip memory**
  - **DDR-3**: 64-bit iface @ 1600 Mhz
- **ECC** for L2 and DDR-3
- **DMA** between memory spaces

First experiences with BLIS on a DSP

BLIS on a DSP: challenges

1. New architecture (VLIW)
2. New compiler (`cl6x`)
3. New operating system (`SYS/BIOS`)
4. New development environment (`Code Composer Studio`)

Result: another BLIS success story

BLIS runs out of the box!

First experiences with BLIS on a DSP

BLIS on a DSP: challenges

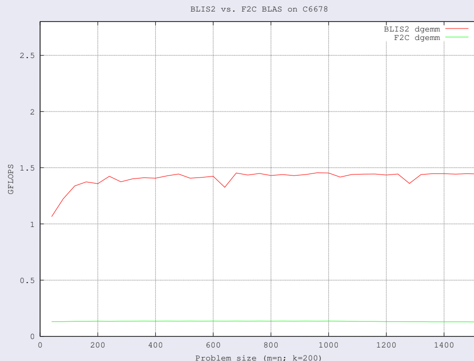
1. New architecture (VLIW)
2. New compiler (`cl6x`)
3. New operating system (`SYS/BIOS`)
4. New development environment (`Code Composer Studio`)

Result: another BLIS success story

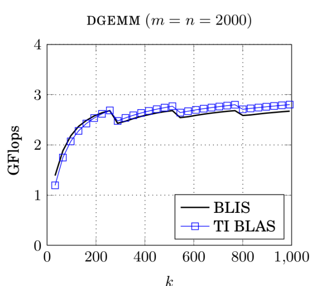
BLIS runs out of the box!

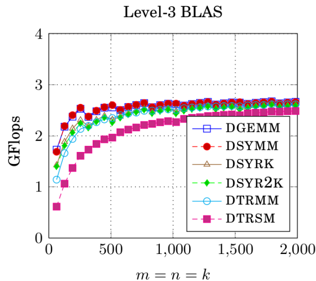
First experiences with BLIS on a DSP

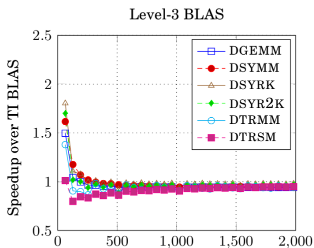
BLIS performance on a single-core DSP: reference BLIS vs. f2c



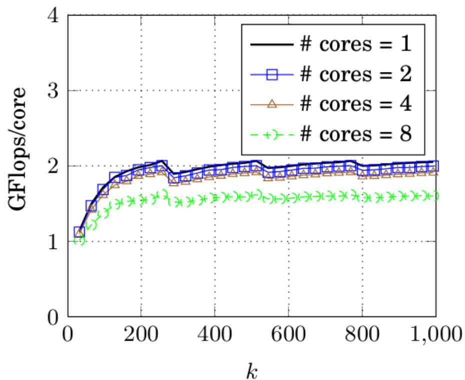
Porting BLIS to C6678 DSP. Sequential results







Porting BLIS to C6678 DSP. Parallel results



First experiences with BLIS on a DSP

Performance analysis and future work

- Reference BLIS version: unoptimized microkernel
  - Unoptimized microkernel
  - No DMA used
- Compared with `f2c` reference DGEMM: 10× improvement
- Attain 60% of highly tuned TI's DGEMM out-of-the-box
- Attain 90%-95% after block size tuning

Ongoing and future work

- Use of optimized microkernels from TI's BLAS on BLIS
- Integration of DMA and scratchpad buffers into BLIS
  - The layered approach of BLIS makes it *easy* to introduce DMA in the framework

Lessons learned

After porting to ARM

- BLIS can be ported to architectures other than x86. BLIS seems portable
- Beat ATLAS for ARM using a microkernel written in C. BLIS is fast
- Fast and cross compilation

After porting to Intel Sandy Bridge

- Beat ATLAS by a wide margin. BLIS is fast
- Beated by OpenBLAS and MKL. BLIS can be faster

After porting to TI C6678

- BLIS compiles and runs using the TI software infrastructure. BLIS is portable
- Attain 90-95% TI BLAS native implementation, without using DMA. BLIS is fast

Outline

1. ARM Cortex A9
2. Intel Sandy Bridge
3. Texas Instruments C6678 DSP
4. Integration of DMA in BLIS
   - Mapping BLIS/GotoBLAS to the C66x DSP core
   - Memory requirements. DMA
5. Conclusions

DMA in BLIS

What we want

- Use DMA to overlap communication and computation
- Easily decide when to use DMA
- Easily decide from which memory level
- Easily decide to which memory level
- With minimal user intervention
- Adapting to different families of DSPs (fully configurable mechanism)

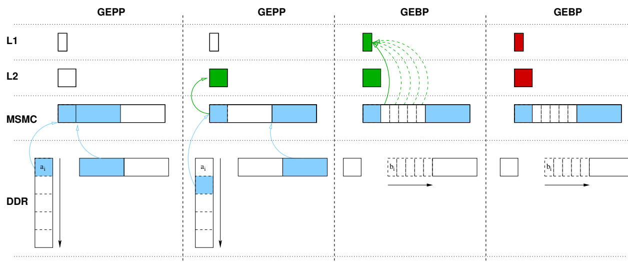
Mapping BLIS/GotoBLAS to the C66x DSP core

- By adapting block sizes, BLIS/GotoBLAS **assume** that blocks of A, B and C will reside in given memory hierarchy levels.
- But on the DSP...
  - L1, L2, MSMC memory can be used as *scratchpad* memories
  - We can **force** blocks/panels of A, B and C to reside in a given hierarchy level during the computation

```
/* Create .myL1 section mapped on L1 cache */
Program.sectMap[ ".myL1" ] = "L1DSRAM";

/* Create .myL2 section mapped on L2 cache */
Program.sectMap[ ".myL2" ] = "L2SRAM";

/* Create .myMSMC section mapped on MSMC */
Program.sectMap[ ".myMSMC" ] = "MSMCSRAM";
```

```
/* L1 allocation */
#pragma DATA_SECTION( pL1, ".myL1" );
float pL1[ L1_SIZE ];

/* L2 allocation */
#pragma DATA_SECTION( pL2, ".myL2" );
float pL2[ L2_SIZE ];

/* MSMC allocation */
#pragma DATA_SECTION( pMSMC, ".myMSMC" );
float pMSMC[ MSMC_SIZE ];
```

Memory requirements and DMA



Overlap computation and communication between memory layers

- Goal: hide overhead of data movements between memory spaces
- Double-buffering: *scratchpad* buffers in three memory levels:

L1	*Packed* sub-block of $\hat{B}$
L2	*Packed* sub-block of $\hat{A}$
MSMC	Buffers to receive *unpacked* buffers of A and sub-panels of $\hat{B}$ from DDR

Integrating DMA in BLIS

Can we integrate this mechanism into BLIS?

DMA integration into BLIS. Required changes:

1. **Memory manager** (Where to DMA?)
2. **Control trees** (When and How to DMA?)

BLIS memory manager

- BLIS defines pools for contiguous memory allocation:
  1. Buffers for blocks of A (`BLIS_BUFFER_FOR_A_BLOCK`, static)
  2. Buffers for panels of B (`BLIS_BUFFER_FOR_B_PANEL`, static)
  3. Buffers for panels of C (`BLIS_BUFFER_FOR_C_PANEL`, static)
  4. Buffers for general use (`BLIS_BUFFER_FOR_GEN_USE`, dynamic)

- These buffers are required as e.g. destination of pack routines
- Configurable at installation time (at `bli_config.h`)

BLIS memory manager. Accommodating DMA

- To accommodate DMA, we can define pools at each level of the memory hierarchy, e.g.
    1. Buffers for blocks of A on L1 (BLIS_BUFFER_FOR_A_BLOCK_L1, static)
    2. Buffers for blocks of A on L2 (BLIS_BUFFER_FOR_A_BLOCK_L2, static)
    3. Buffers for blocks of A on L3 (BLIS_BUFFER_FOR_A_BLOCK_L3, static)
    4. ...
    5. Buffers for general use on L1 (BLIS_BUFFER_FOR_GEN_USE_L1, dynamic)
    6. Buffers for general use on L2 (BLIS_BUFFER_FOR_GEN_USE_L1, dynamic)
    7. Buffers for general use on L3 (BLIS_BUFFER_FOR_GEN_USE_L3, dynamic)
- These buffers are required as e.g. destination of pack or DMA routines
- We fix each pool at a given level of the memory hierarchy level
- Configurable at installation time (at bli_config.h)

BLIS memory manager. Accommodating DMA

- Full flexibility to adapt to different product families, with different cache configurations
- Full flexibility to decide origin and destination level of packing and DMA routines

Do we need to rewrite the internal implementations to, e.g.

- DMA block of A from DDR to MSMC, and pack it from MSMC to L2? Or...
- DMA block of A from DDR to L2, and pack it from L2 to L1? Or...
- DMA block of A from DDR to MSMC, and pack it from MSMC to MSMC? Or...
- ...
- We don't want DMA?
- ...

Complicated!!

BLIS memory manager. Accommodating DMA

- Full flexibility to adapt to different product families, with different cache configurations
- Full flexibility to decide origin and destination level of packing and DMA routines

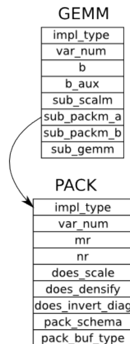
Do we need to rewrite the internal implementations to, e.g.

- DMA block of A from DDR to MSMC, and pack it from MSMC to L2? Or...
- DMA block of A from DDR to L2, and pack it from L2 to L1? Or...
- DMA block of A from DDR to MSMC, and pack it from MSMC to MSMC? Or...
- ...
- We don't want DMA?
- ...

Complicated!!

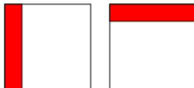
Control trees (a.k.a. Field's magic glue)

- Control trees: mechanism to encode algorithmic control information **a priori**
- Passed and processed by internal implementations - Hidden to the user
- Default c.t. are selected when invoking the front-ends
- BLIS provides an API to create and manage c.t.
- Thus, the developer can tune the algorithmic behavior of the operation

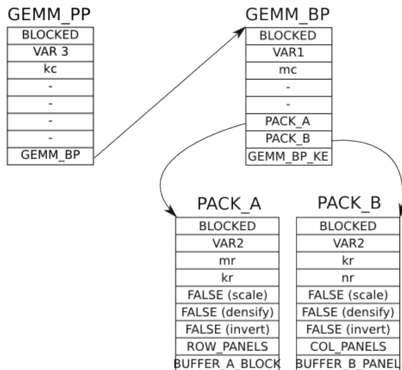


Control trees (a.k.a. Field's magic glue)

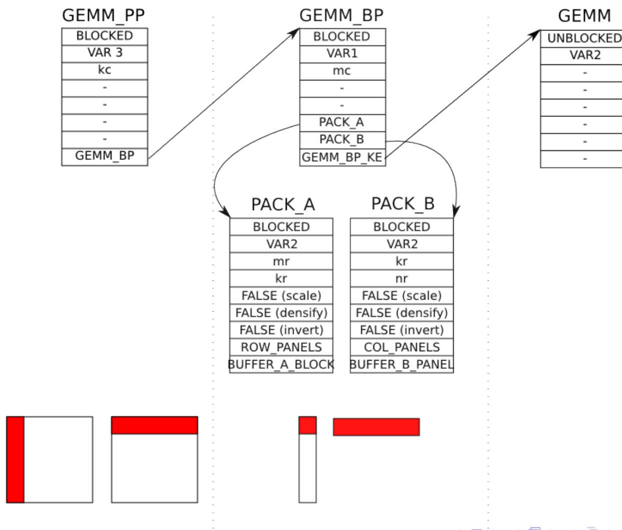
GEMM_PP
BLOCKED
VAR 3
kc
-
-
-
-
GEMM_BP



Control trees (a.k.a. Field's magic glue)

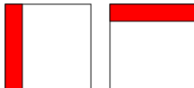


Control trees (a.k.a. Field's magic glue)

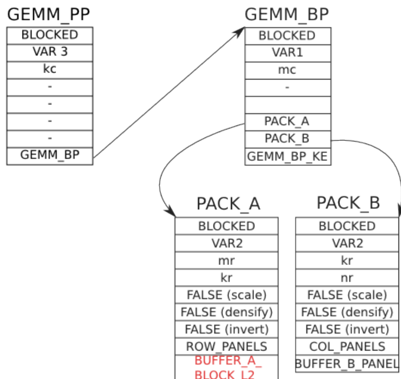


Control trees (a.k.a. Field's magic glue)

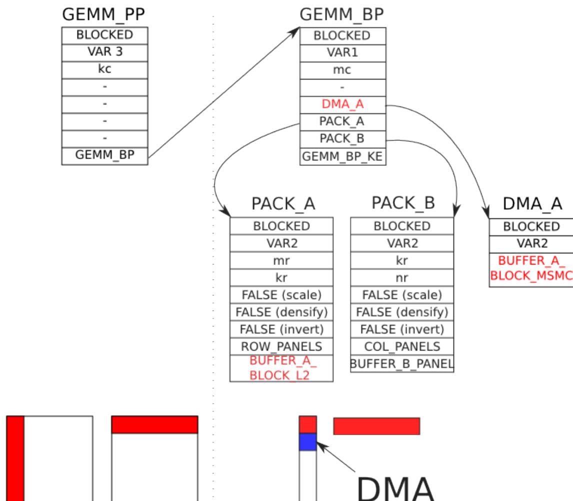
GEMM_PP
BLOCKED
VAR 3
kc
-
-
-
-
GEMM_BP



Control trees (a.k.a. Field's magic glue)



Control trees (a.k.a. Field's magic glue)



Control trees (a.k.a. Field's magic glue)

With these modifications, the developer (vendor) can:

- Integrate DMA into the framework
- Easily evaluate the benefits of using DMA at a given point
- Easily manage scratchpad buffers
- Fully exploit memory hierarchy
- Adapt the DMA mechanism to the specific architecture without modifying BLIS internals

Not studied yet:

- How to adapt this mechanism to other architectures with DMA, e.g. Parallella

Outline

1. ARM Cortex A9
2. Intel Sandy Bridge
3. Texas Instruments C6678 DSP
4. Integration of DMA in BLIS
   - Mapping BLIS/GotoBLAS to the C66x DSP core
   - Memory requirements. DMA
5. Conclusions

Conclusions

- BLIS is **portable** (tested on AMD A10, Power7, Power A2, Xeon Phi, Loongson 3A)
- BLIS is **fast**. Very competitive performance in all architectures tested
- BLIS is **extensible**. DMA mechanism can be easily integrated
- BLIS is **stable**. No big bugs despite alpha versions

Thanks for your attention!

Francisco D. Igual (`figual@ucm.es`)