TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing

Identification of Performance Problems

Ulf Markwardt Ulf.Markwardt@tu-dresden.de

Special thanks to:

Heike Jagode jagode@eecs.utk.edu
Dan Terpstra terpstra@eecs.utk.edu

VI-HPS

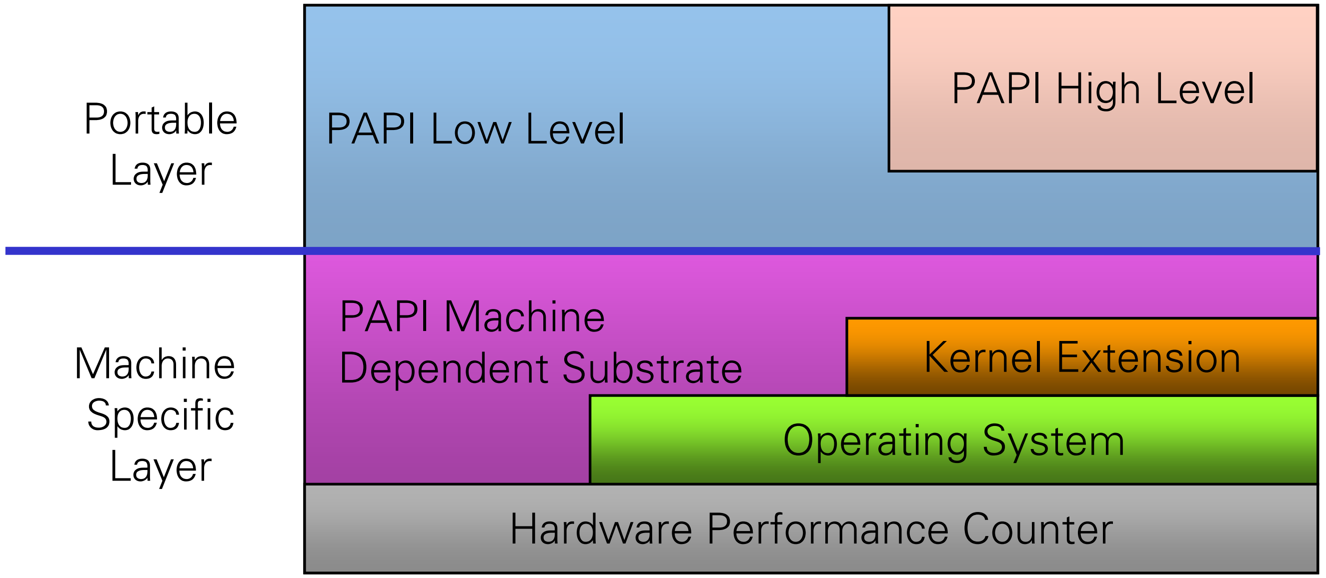
ZIH Center for Information Services & High Performance Computing

Contents

What is PAPI ?

Performance **A**pplication **P**rogramming **I**nterface for hardware performance counters

- http://icl.cs.utk.edu/projects/papi
- Started as a Parallel Tools Consortium project in 1998
- Goal was to produce a specification for a portable interface to the hardware performance counters



PAPI tools – papi_cost

```
mark@n13:~> papi_cost -b 5 -d
Cost of execution for PAPI start/stop and PAPI read.
This test takes a while. Please be patient...
Performing start/stop test...

Total cost for PAPI_start/stop(2 counters) over 1000000 iterations
min cycles   : 6389
max cycles   : 48555741
mean cycles  : 6910.000000
std deviation: 50860.901567

Cost distribution profile

    6389:**************************** 999998 counts ****************************
 9716259:****************************************************************************************************
19426129:
29135999:
38845869:


Performing read test...

Total cost for PAPI_read(2 counters) over 1000000 iterations
min cycles   : 112
max cycles   : 21150
mean cycles  : 117.000000
std deviation: 36.541767

Cost distribution profile

     112:**************************** 999995 counts ****************************
    4319:****************************************************************************************************
    8526:**************************************************
   12733:**************************************************
   16940:
```

PAPI tools – papi_mem_info

```
mark@n13:~> papi_mem_info
Test case:  Memory Information.
------------------------------------------------------------------------
L1 Instruction TLB:  Number of Entries: 512;  Associativity: 4

L1 Data TLB:  Number of Entries: 512;  Associativity: 4

L1 Instruction Cache:
  Total size: 64KB
  Line size: 64B
  Number of Lines: 1024
  Associativity: 2

  Total size: 64KB
  Line size: 64B
  Number of Lines: 1024
  Associativity: 2

  Total size: 1024KB
  Line size: 64B
  Number of Lines: 16384
  Associativity: 16
```

PAPI tools – papi_avail

```
mark@n13:~> papi_avail -h
This is the PAPI avail program.
It provides availability and detail information
for PAPI preset and native events.  Usage:

    avail [options] [event name]
    avail TESTS_QUIET

Options:

  -a            display only available PAPI preset events
  -d            display PAPI preset event info in detailed format
  -e EVENTNAME  display full detail for named preset or native event
  -h            print this help message
  -t            display PAPI preset event info in tabular format (default)
```

PAPI tools – papi_avail

```
mark@n13:~> papi_avail -a
Test case avail.c: Available events and hardware information.
-------------------------------------------------------------------------
Vendor string and code   : AuthenticAMD (2)
Model string and code    : AMD K8 Revision C (15)
CPU Revision             : 10.000000
CPU Megahertz            : 2205.074951
CPU's in this Node       : 2
Nodes in this System     : 1
Total CPU's              : 2
Number Hardware Counters : 4
Max Multiplex Counters   : 32
-------------------------------------------------------------------------
Name            Derived Description (Mgr. Note)
PAPI_L1_DCM     Yes     Level 1 data cache misses ()
PAPI_L1_ICM     Yes     Level 1 instruction cache misses ()
PAPI_L2_DCM     No      Level 2 data cache misses ()
PAPI_L2_ICM     No      Level 2 instruction cache misses ()
PAPI_L1_TCM     Yes     Level 1 cache misses ()
PAPI_L2_TCM     Yes     Level 2 cache misses ()
PAPI_FPU_IDL    No      Cycles floating point uni
PAPI_TLB_DM     No      Data translation lookasic
[…]
```

TODO papi_avail:

- Check performance counters available on the system
- Get acquainted with preset counters

PAPI tools – papi_native_avail

```
mark@n13:~> papi_native_avail
[…]
The following correspond to fields in the PAPI_event_info_t structure.
Symbol  Event Code      Count
 |Short Description|
 |Long Description|
 |Derived|
 |PostFix|
The count field indicates whether it is a) available (count >= 1) and b) derived
(count > 1)
FP_ADD_PIPE     0x40000000
 |Dispatched FPU ops - Revision B and later revisions - Speculative add pipe ops
excluding junk ops|
 |Register Value[0]: 0xf          P3 Ctr Mask|
 |Register Value[1]: 0x100        @?|

FP_MULT_PIPE    0x40000001
 |Dispatched FPU ops - Revision B and later revisions - Speculative multiply pipe
ops excluding junk ops|
 |Register Value[0]: 0xf          P3 Ctr Mask|
 |Register Value[1]: 0x200        @?|
[…]
```

TODO papi_native_avail:

- Difference between native and PAPI counters

PAPI tools – papi_event_chooser

```
mark@n13:~> papi_native_avail PRESET FP_ADD_PIPE PAPI_FP_INS FR_X86_INS
Test case eventChooser: Available events which can be added with given events.
[…]
Name              Derived Description (Mgr. Note)
PAPI_L2_DCM       No      Level 2 data cache misses ()
PAPI_L2_ICM       No      Level 2 instruction cache misses ()
PAPI_FPU_IDL      No      Cycles floating point units are idle ()
PAPI_TLB_DM       No      Data translation lookaside buffer misses ()
PAPI_TLB_IM       No      Instruction translation lookaside buffer misses ()
PAPI_L1_LDM       No      Level 1 load misses ()
PAPI_L1_STM       No      Level 1 store misses ()
PAPI_L2_LDM       No      Level 2 load misses ()
PAPI_L2_STM       No      Level 2 store misses ()
PAPI_STL_ICY      No      Cycles with no instruction issue ()
PAPI_HW_INT       No      Hardware interrupts ()
PAPI_BR_TKN       No      Conditional branch instructions taken ()
PAPI_BR_MSP       No      Conditional branch instructions mispredicted ()
PAPI_TOT_INS      No      Instructions completed ()
PAPI_BR_INS       No      Branch instructions ()
PAPI_VEC_INS      No      Vector/SIMD instructions ()
PAPI_RES_STL      No      Cycles stalled on any resource ()
PAPI_TOT_CYC      No      Total cycles ()
PAPI_L2_DCH       No      Level 2 data cache hits ()
```

PAPI tools – papi_command_line

```
mark@n13:~> papi_command_line FP_ADD_PIPE PAPI_FP_INS FR_X86_INS PAPI_TOT_CYC
[…]
This utility lets you add events from the command line interface to see if they
work.
command_line.c                                   PASSED
mark@n13:~> papi_command_line FP_ADD_PIPE PAPI_FP_INS FR_X86_INS PAPI_VEC_INS
PAPI_TOT_CYC
Successfully added: FP_ADD_PIPE
Successfully added: PAPI_FP_INS
Successfully added: FR_X86_INS
Successfully added: PAPI_VEC_INS
Failed adding: PAPI_TOT_CYC
because: PAPI_ECNFLCT

FP_ADD_PIPE :   20001228
PAPI_FP_INS :   40002472
FR_X86_INS :    220164398
PAPI_VEC_INS :  0
PAPI_TOT_CYC :  ---------

----------------------------------
Verification: None.
 This utility lets you add events from the commar
work.
```

TODO papi_event_chooser, papi_command_line

- Which events can be counted simutaneously?
- Sure?

PAPI tools

Check PAPI utilities at Phobos, get a feeling what can be analyzed!

```
~> module load papi
```

- Memory hierachy
  papi_mem_info
- Costs of PAPI calls
  papi_cost
- Available native/derived/preset counters
  papi_avail , **papi_native_avail**
- Combinable counters
  papi_event_chooser
- Check out to dampen you anticipation
  papi_command_line

PAPI access from C or Fortran

Countable events are defined in two ways:

- Platform-neutral **preset** events (e.g., PAPI_TOT_INS)
- Platform-dependent **native** events (e.g., L3_CACHE_MISS)

Additionally, multiplex measurements can be defined – for a mere statistical analysis.

Levels of access

- High level API calls makes access to preset counters easier
- Low level API calls can access all available counters for a few lines more (or wrapper functions)

PAPI high level functions

PAPI_num_counters - get the number of hardware counters available on the system

PAPI_flips - simplified call to get Mflips/s (floating point instruction rate), real and processor time

PAPI_flops - simplified call to get Mflops/s (floating point operation rate), real and processor time

PAPI_ipc - gets instructions per cycle, real and processor time

PAPI_accum_counters - add current counts to array and reset counters

PAPI_read_counters - copy current counts to array and reset counters

PAPI_start_counters - start counting hardware events

PAPI_stop_counters - stop counters and return current counts

- Easy to use functions - but works only for a limited number (42 - sic!) pre-defined events.
- Get preset events with: **`papi_avail -a`**

PAPI low level functions

PAPI_accum (3) - accumulate and reset hardware events from an event set
PAPI_add_event (3) - add single PAPI preset or native hardware event to an event set
PAPI_add_events (3) - add array of PAPI preset or native hardware events to an event set
PAPI_attach (3) - attach specified event set to a specific process or thread id
PAPI_cleanup_eventset (3) - remove all PAPI events from an event set
PAPI_create_eventset (3) - create a new empty PAPI event set
PAPI_destroy_eventset (3) - deallocates memory associated with an empty PAPI event set
[…]

Total of 74 functions covering the whole functionality of the PAPI library. See e.g. **http://icl.cs.utk.edu/projects/papi/files/html_man3/papi.html** for reference.

For this practical, wrappers (see "papi" directory) for low level functions are provided:

- **`initPAPI(char* sList[])`** – init PAPI, add events from event list, start counters
- **`stopPAPI()`** – stop hardware counters
- **`readPAPIcounters(long long values[])`** – read counters.
  User has to define array!

Performance measurement categories

Efficiency

- Retired instructions per cycle (PAPI_TOT_INS / PAPI_TOT_CYC)

Caches

- Data cache misses and miss ratio
- Cache misses and miss ratio

Translation lookaside buffers (TLB)

- Data TLB misses and miss ratio
- Instruction TLB misses and miss ratio

Floating point operations (# integer ops)

- Retired, stalled, speculative
- exceptions

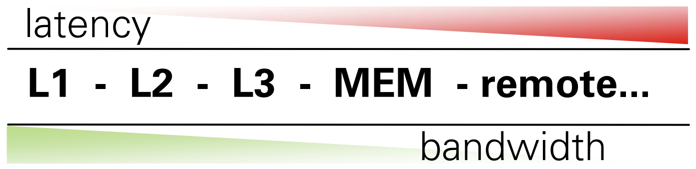
Control structures

- Branch mispredictions

Typical performance problems

- Code has superfluous sections? Re-design it!
- All computations are necessary! - Decrease the time in between. Avoid stalled pipelines.

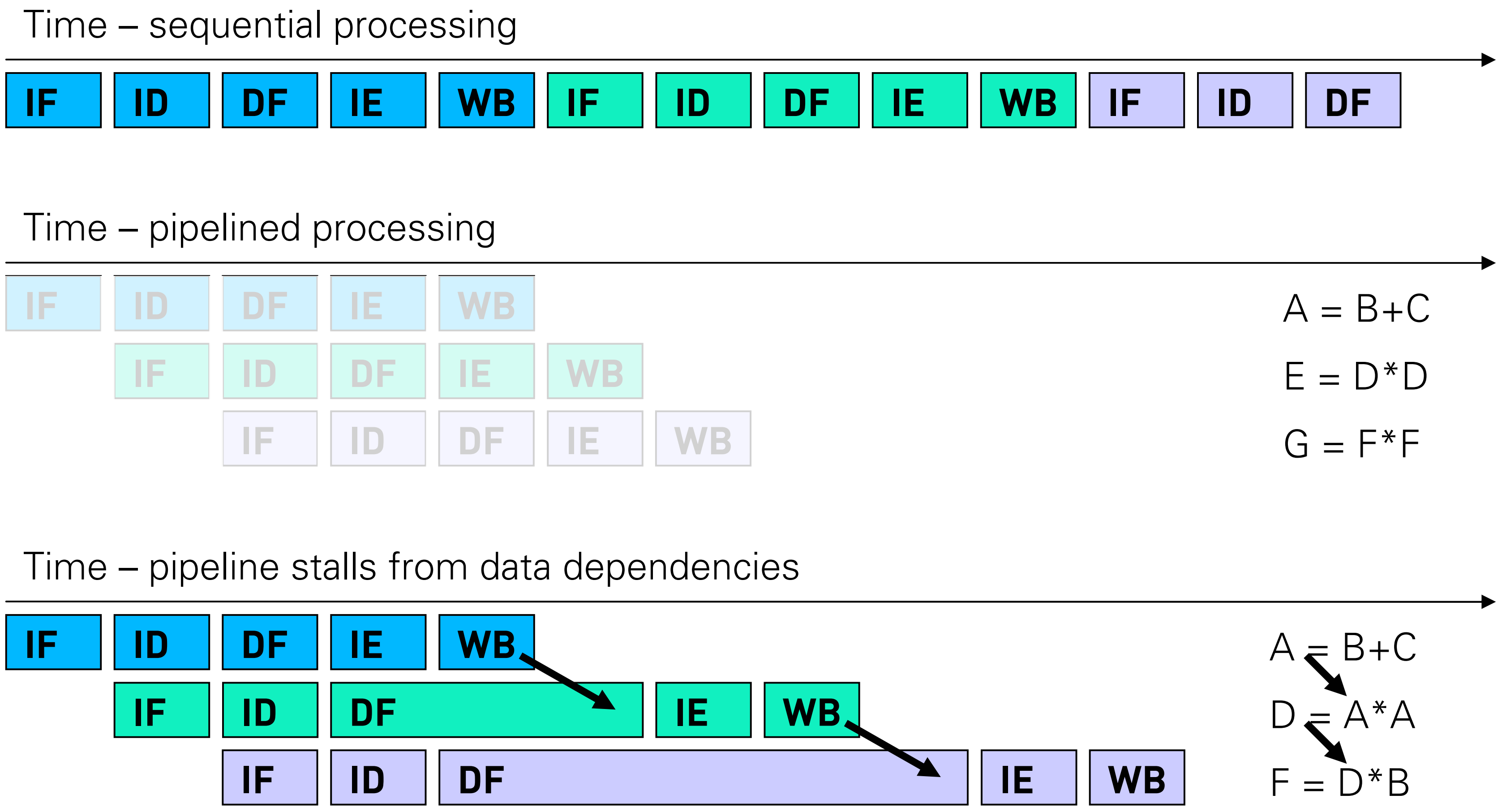
1 **Memory wall**: growing disparity of speed between CPU and memory outside the CPU chip.



- Even streaming memory bandwidth is too slow for super-pipelined arithmetic units.
- Latency for first reference.
- TLB misses make things worse.

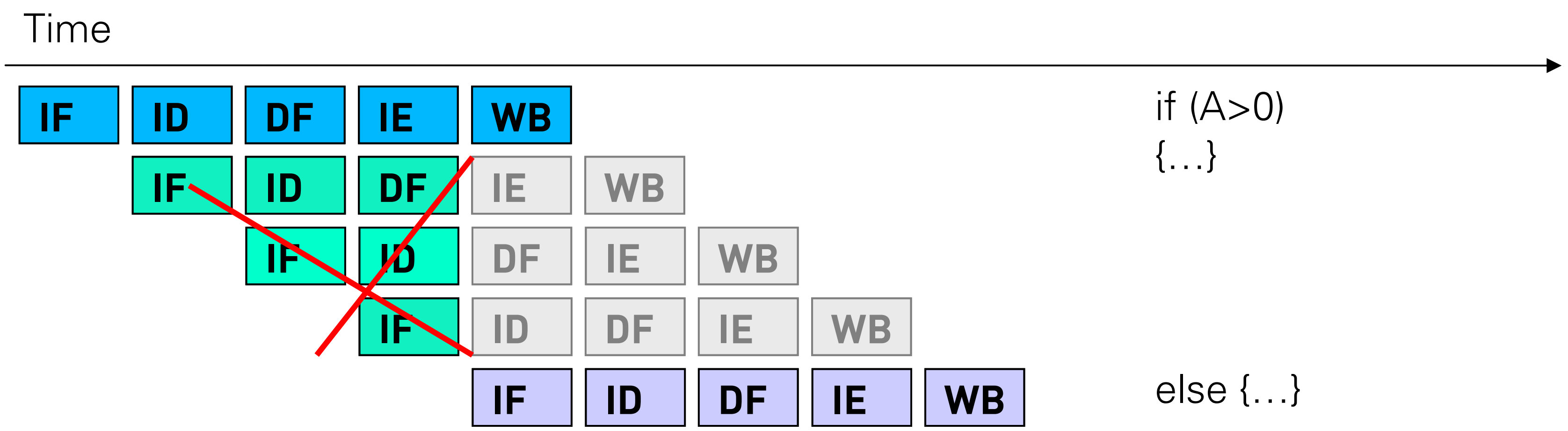
Typical performance problems

2 Data Dependencies: Too near data dependencies can stall pipes


Time – sequential processing
IF ID DF IE WB IF ID DF IE WB IF ID DF
Time – pipelined processing
IF ID DF IE WB
IF ID DF IE WB
IF ID DF IE WB
A = B+C
E = D*D
G = F*F
Time – pipeline stalls from data dependencies
IF ID DF IE WB
IF ID DF IE WB
IF ID DF IE WB
A = B+C
D = A*A
F = D*B

Typical performance problems

3 **Branch mispredictions:** Invalidate pipeline



4 **Data alignemt:** SSE expects alinged data

In general: Help the compiler to optimize the code

Memory wall – data cache access

Cache miss: a failed attempt to read or write a piece of data in the cache

- Results in main memory access with much longer latency
- Important to keep data as close as possible to CPU

Classes of data cache misses

- **Compulsory misses**: first reference to a data item
  - Prefetching can help
- **Capacity misses**: working set exceeds the cache capacity
  - Spatial locality: use all the data that is loaded into the cache
  - Smaller working set (blocking/tiling algorithms)
- **Conflict misses**: data item is referenced after its cache line was evicted earlier.
  - Temporal locality: reuse data as long as possible
  - Data layout; memory access patterns

Code optimization – data structure

- Focus on spatial and temporal data locality!
- Reuse data as much as possible (even at the price of more computations)
- Always: Compromise between performance and readability:
  preprocessor macros may help without performance degradation

- Example: calculate $\sum \|(x,y,z)^T\|$ for given data
  - Compund structure
    - `typedef struct {`
    - `  char            valid; /* just a tag */`
    - `  PRECISION  x, y, z;`
    - `} R3 ;`
  - Each dimension in a seperate vector:
    `PRECISION x[N], y[N], z[N];`

TODO data_structure:

•Use different data types, structures

•Check L1/L2 cache misses, FLOPS

•Instructions per cycle

Code optimization – data structure

```
PRECISION do_flops(PRECISION x[],PRECISION  y[],PRECISION  z[])
{
 int i;
 PRECISION s=0;
 for (i=0; i< N; i++) {
     s = s + x(i)*x(i) + y(i)*y(i) + z(i)*z(i);
 }
 return sqrt(s);
}
```

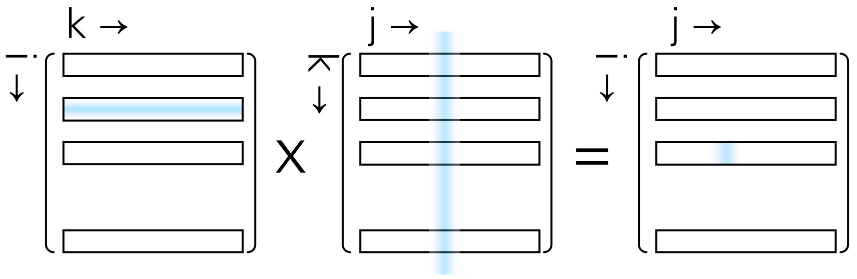
- Data locality (in a struct) clearly wins this competition
  - Access one data stream instead of three
- Observe automatic padding of struct (with and without __attribute__((packed)) and its influence on performance
  - Data alignment is critical for SSE instructions

Code optimization – data access

Example 2: Matrix multiplication

```
a=(PRECISION*)
   (malloc(N *N* sizeof(PRECISION)) );
b= ...

PRECISION do_flops(
   PRECISION a[],
   PRECISION b[],
   PRECISION c[])
{
   int i,j,k;double s;
   for (i=0; i<N; i++) {
      for (j=0; j<N; j++) {
         for (k=0; k<N; k++) {
            c[i*N+j]+=
   a[i*N+k]*b[k*N+j];
         }
      }
    }
   return 0.0;
}
```



TODO example_1:

•Use different data types, structures

•Check L1/L2 cache misses, FLOPS

•Instructions per cycle

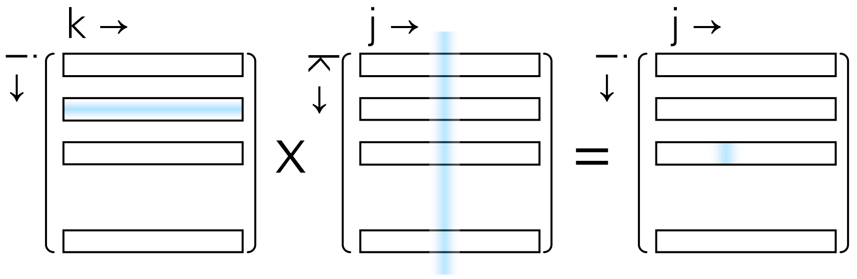
•Find better solution?

Code optimization – data access

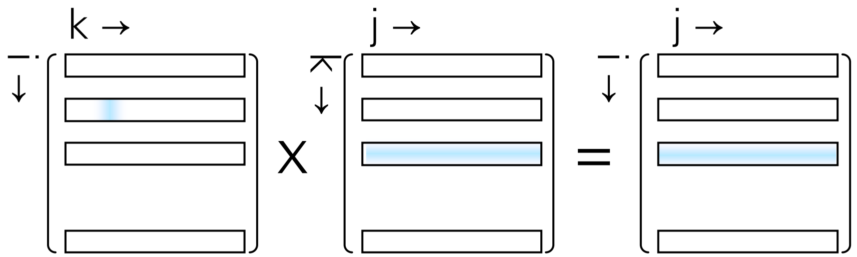
Example 2: Matrix multiplication

```
a=(PRECISION*)
   (malloc(N *N* sizeof(PRECISION)) );
b= ...

PRECISION do_flops(
   PRECISION a[],
   PRECISION b[],
   PRECISION c[])
{
   int i,j,k;double s;
   for (i=0; i<N; i++) {
      for (k=0; k<N; k++) {
      for (j=0; j<N; j++) {
         {
            c[i*N+j]+=
   a[i*N+k]*b[k*N+j];
         }
      }
    }
   return 0.0;
}
```



Loop rearranging leads to faster data access along two cache lines



TODO example_1:

•Compare measurements

Cheating allowed

- High level programming languages are already a compromise between usability and performance
- For standard problems: always check for an appropriate library first
  - Hardware optimized libraries (ACML, MKL)
  - Community libraries (ATLAS, SCALAPACK, GSL)
  - Commercial products (NAG, IMSL)
    Attention: May be unusable without prior warning!

TODO example_2:

•Compare measurements with MKL and ACML

•Use single precision for floating point data,
check PAPI_VEC_INS

Comparison of measurements

	ijk	ikj	ACML	MKL
Time (s)	14,49	4,60	0,55	0,65
PAPI_TOT_CYC	31449267176	9903957877	1155592270	1392980080
PAPI_TOT_INS	11007032168	11009023783	1630767579	1708966373
instr/cyc	0,350	1,112	1,411	1,227
PAPI_L1_DCH				
PAPI_L1_DCM				
L1 hit ratio				
PAPI_L2_DCH				
PAPI_L2_DCM				
L2 hit ratio				

Code optimizations

Scope: Reduce stalls in the pipelines of the arithmetic units

- Increase cache hits (low latency, higher bandwidth)
  - Avoid „random“ (i.e. non-linear) access to data
- Avoid branch mispredictions
  - Complicated loops (compiler has no clue)
  - If-conditions in loops, premature loop exits
  - Function calls (without inlining)
- Data alignment
- Avoid immediate data dependencies

- Always check effect of code modifications.
- Sometimes compilers think they are smart.