JÜLICH
FORSCHUNGSZENTRUM

JUQUEEN

Best Practices

4. Februar 2013 | Florian Janetzko

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Outline

Production Environment

- Module Environment

- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Module Environment

Module concept

- Provides overview over available software packages
- Eases use of software packages
  - Access to software packages, libraries
  - Supply of different versions of applications
  - Supply of application-specific information
- Enables dynamic modification of users' environment
  - Environment variables (**`PATH`**, **`LD_LIBRARY_PATH`**, **`MANPATH`**, …) are set appropriately
  - Detection of conflicts between applications

Module Environment

```
$ module <options> <module>
```

Option	Description
`<no option>`	Lists available options of the module command
`avail`	Lists all available modules
`list`	Lists modules currently loaded
`load`	Loads a module
`unload`	Unloads a module
`help`	Lists information about a module
`show`	Information about settings done by the module
`purge`	Unloads all modules

Module Environment

Six module categories

- COMPILER
  - *Different compilers and versions of compilers*
- IO
  - *I/O libraries and tools*
- MATH
  - *Mathematical libraries and software packages*
- MISC
  - *Software not fitting into another category*
- SCIENTIFIC
  - *Software packages from different scientific fields*
- TOOLS
  - *Performance analysis, debugger, etc.*

Software for

Compute Nodes:

`/bgsys/local`

Front-end Nodes:

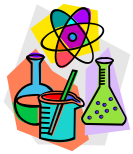
`/usr/local`

Module Environment – Applications & Libraries

Mathematical applications and libraries

arpack (2.1)	gsl (1.15)	mumps (4.10.0)	scalapack (2.0.1)
fftw (2.1.5,3.3.2)	hypre (2.8.0)	parmetis (3.2.0,4.0.2)	sprng (1.0, 2.0)
gmp (5.0.5)	lapack (3.3.0)	petsc (3.3)	sundials (2.5.0)

Scientific applications



CPMD (3.15.1)	Gromacs (4.5.5)	OpenFOAM*
CP2K (2.2.12394)	Lammps (5May12,30Aug12)	QuantumEspresso*
GPAW*	Namd (2.8, 2.9)	VASP**

* In preparation

** Software not installed but makefiles are available

Module Environment – Applications & Libraries

I/O libraries

HDF5***

netCDF***

SIONlib***

→Talk:
Wolfgang Frings, JSC
Parallel I/O
5. February 10:15

Tools

Cmake (2.8.8)

hpctoolkit (5.2.1)

Tau (2.21.2, 2.21.3)

DDT*

PAPI (4.4.0)

Totalview (8.11.0)

Extrae (2.2.1)

Scalasca (1.4.2)

Darshan (2.2.4)

→Talk:
Markus Geimer, JSC
Performance tools and debuggers
4. February 11:00

* In preparation

*** Software installed, module files to be installed

Outline

Production Environment

- Module Environment
- Job Execution 

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

LoadLeveler Batch System – Commands

Execution of applications managed by LoadLeveler

http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/LoadLeveler.html

Command	Description
`llsubmit <jobfile>`	Sends job to the queuing system

`llq`
`llq -l <job ID>`
`llq -s <job ID>`
 	
`llq -u <user>`
	Lists all queued and running jobs
	detailed information about the specified job
	detailed information about a specific queued job, e.g. expected start time
	lists all jobs of the specified user

`llcancel <job ID>`	Kills the specified job
`llstatus`	Displays the status of LoadLeveler
`llclass`	Lists existing classes and their properties
`llqx`	Shows detailed information about all jobs

LoadLeveler – Job Command File

ASCII file containing two major parts

1. LoadLeveler job keywords block at the beginning of a file
   - LoadLeveler keywords have the form
     `#@<keyword>`
   - # and @ can be separated by any number of blanks

2. One or more application script blocks
   - Regular shell script
   - Can contain any shell command

LoadLeveler – Standard Keywords

Keyword	Description
`#@job_name=<name>`	Name of the job

`#@notification=`
`end`
`error`
`never`
`start`
`always`
	Send notification
	if the job is finished
	if the job returned an error code ≠ 0
	never
	upon the start of the job
	combination of `start, end, error`

`#@notify_use=<mailaddr>`	Mail address to send messages to
`#@wall_clock_limit=hh:mm:ss`	Requested wall time for the job

`#@input=<input file name>`
`#@output=<file name for stdout>`
`#@error=<file name for stderr>`
	Specifies corresponding file names

`#@environment=[<variable>,` COPY_ALL]	Environment variable to be exported to job
`#@queue`	Queue job

LoadLeveler – Blue Gene/Q Keywords

Keyword	Description
`#@job_type=[serial, bluegene]`	Specifies the type of job step to process. Must be set to **`bluegene`** for parallel applications.
`#@bg_size=<number of nodes>`	Size of the Blue Gene job, keywords **`bg_size`** and **`bg_shape`** are mutually exclusive.
`#@bg_shape=<A>x<B>x<C>x<D>`	Specifies the requested shape of a job. The max. shape on JUQUEEN is 2x2x2x2.
`#@bg_rotate=[True,False]`	whether the scheduler should consider all possible rotations of the given shape
`#@bg_connectivity=[TORUS,MESH,EITHER] Xa Xb Xc Xd`	Type of wiring requested for the block (can be specified for each dimension separately)

LoadLeveler – Job Classes

Class name	#Nodes	Max. run time	Default run time
n001	1 – **32**	00:30:00	00:30:00
n002	33 – **64**	00:30:00	00:30:00
n004	65 – **128**	12:00:00	06:00:00
n008	129 – **256**	12:00:00	06:00:00
m001	257 – **512**	24:00:00	06:00:00
m002	513 – **1024**	24:00:00	06:00:00
m004	1025 – **2048**	24:00:00	06:00:00
m008	2049 – **4096**	24:00:00	06:00:00
m016	4097 – **8192**	24:00:00	06:00:00
m032*	8193 – **16384**	24:00:00	06:00:00
m048*	16385 – **24576**	24:00:00	06:00:00
m056*	**24577 –** **28672**	24:00:00	06:00:00

*On demand only

You will be charged for the full partition (e.g. if you request 513 nodes you will be charged for 1024 nodes!) → Always use full partitions!

LoadLeveler – Job Scheduling

Backfill scheduler

- The biggest job has the highest priority (*Top Dog*)
- LoadLeveler fills gaps with smaller, short-running jobs while freeing the system for the Top Dog

Tip: Specify the wall time for your jobs as exact as possible, because jobs requesting a shorter wall time have a better chance to be executed.

Big jobs

- Jobs requesting >16 racks are collected and run in dedicated time slots (e.g. after a maintenance) at least once a week

Running Simulations – runjob Command

Launch command for parallel applications

```
runjob [options]
runjob [options]: <executable> [arguments]
```

Option	Description
`--args <prg_arg>`	Passes "**prg_arg**" to the launched application on the compute node.
`--exe <executable>`	Specifies the full path to the executable
`--exp <ENV_Var=Value>`	Sets the environment variable **ENV_Var=Value**
`--exp-env <ENV_Var>`	Sets the environment variable **ENV_Var**
`--np <number>`	Total number of (MPI) tasks
`--ranks-per-node <number>`	Number of (MPI) tasks per compute node

Running Simulations – MPI/OpenMP Codes

- On Blue Gene/P
  - Three modes were available
    1. VN mode (4 MPI tasks, no thread per task)
    2. DUAL mode (2 MPI tasks with 2 OpenMP threads each)
    3. SMP mode (1 MPI task with 4 OpenMP threads)
- On Blue Gene/Q
  - One node has 16 cores with 4-way SMT each
  - Several configurations possible
    - $\texttt{ntasks} \times \texttt{nthreads} = 64$
    - $\texttt{ntasks} = 2^{n},\ 0 \leq n \leq 6$

Test carefully, which configuration gives the best performance for your application and setup!

LoadLeveler – Example Job Command File

```
#@job_name          = hybrid_code
#@comment           = "16x4 configuration"
#@output            = test_$(jobid)_$(stepid).out
#@error             = test_$(jobid)_$(stepid).err
#@environment       = COPY_ALL
#@job_type          = bluegene
#@notification      = never
#@bg_size           = 512
#@bg_connectivity   = torus
#@wall_clock_limit = 14:00:00
#@queue

runjob --np 8192 --ranks-per-node 16\
       --env OMP_NUM_THREADS=4 --exe app.x
```

Filesystems

$HOME (*group* limits: 6 TB, 2 million files)

- for source code, binaries, libraries and applications
- Automatic backup

$WORK (*group* limits: 20 TB, 4 million files)

- temporary storage for applications
- No automatic backup, files older than 90 days are deleted automatically

$ARCH (group limits: 2 million files)

- Storage of data during the project's lifetime
- Use file archives (tar), store files of size 500 – 1000 GB

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Compilers

- Different compilers for front-end and compute nodes
- GNU and IBM XL family of compilers available

Tip: It is recommended to use the XL suite of compilers for the CN since they produce in general better optimized code.

FRONT-END

Language	XL compiler	GNU compiler
C	`xlc, xlc_r`	`gcc`
C++	`xlc++, xlc++_r, xlC, xlC_r`	`g++`
Fortran	`xlf, xlf90, xlf95, xlf2003` `xlf_r, xlf90_r, xlf95_r, xlf2003_r`	`gfortran`

Compilers for CN

GNU (GCC)

Language	Compiler invocation	MPI wrapper
C	`powerpc64-bgq-linux-gcc`	`mpigcc`
C++	`powerpc64-bgq-linux-g++`	`mpig++`
Fortran	`powerpc64-bgq-linux-gfortran`	`mpigfortran`

XL COMPILERS

Language	Compiler invocation (thread-safe: *_r)	MPI wrapper (thread-safe: *_r)
C	`bgxlc, bgc89, bgc99`	`mpixlc`
C++	`bgxlc++, bgxlC`	`mpixlcxx`
Fortran	`bgxlf, bgxlf90, bgxlf95, bgxlf2003`	`mpixlf77, mpixlf90, mpixlf95, mpixlf2003`

Basic Compiler Options – XL Compilers I

Flags in order of increasing optimization potential

Optimization Level	Description
`-O2 -qarch=qp -qtune=qp`	Basic optimization
`-O3 -qstrict -qarch=qp -qtune=qp`	More aggressive, not impact on acc.
`-O3 -qhot -qarch=qp -qtune=qp`	More aggressive, may influence acc. (high-order transformations of loops)
`-O4 -qarch=qp -qtune=qp`	Interprocedural optimization at compile time
`-O5 -qarch=qp -qtune=qp`	Interprocedural optimization at link time, whole program analysis

Basic Compiler Options – XL Compilers II

Additional compiler flags

Compiler Flag	Description
`-qsmp=omp -qthreaded`	Switch on OpenMP support
`-qreport -qlist`	Generates for each source file **`<name>`** a file **`<name>.lst`** with pseudo code and a description of the kind of code optimizations which were performed
`-qessl –lessl[smp]bg`	Compiler attempts to replace some intrinsic FORTRAN 90 procedures by essl routines where it is safe to do so

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Diagnostic Compiler Flags (XL Compilers)

Diagnostic messages are given on the terminal and/or in a separate file

- **`-qreport`**: compilers generate a file **`name.lst`** for each source file
- **`-qlist`**: compiler listing including an object listing
- **`-qlistopt`**: options in effect during compilation included in listing

Listen to the compiler!

- **`-qflag=<listing-severety>:<terminal-severety>`**
  - *i: informal messages, w: warning messages, s: severe errors*
  - *Use* ***`-qflag=i:i`*** *to get all information*
- **`-qlistfmt=(xml|html)=<option>`**

Example: Compilers Diagnostics

```
subroutine mult(c,a,ndim)

implicit none
integer :: ndim,i,j
double precision ::
a(ndim),c(ndim,ndim)

! Loop
do i=1,1000
  do j=1,1000
    c(i,j) = a(i)
  enddo
enddo

end subroutine mult
```

```
>>>>> LOOP TRANSFORMATION SECTION <<<<<
1| SUBROUTINE mult (c, a, ndim)

[...]
Id=1   DO $$CIV2 = $$CIV2,124
   10|   IF (.FALSE.) GOTO lab_11
           $$LoopIV1 = 0
Id=2   DO $$LoopIV1 = $$LoopIV1,999
[...]
-----------------------------------
0 9 1    Loop interchanging applied
         to loop nest.
0 9 1    Outer loop has been
         unrolled 8 time(s).
```

Single-Core Optimization – Compiler Flags

Take advantage of vector instructions

`-qsimd=auto`

Function inlining

`-qinline=auto:level=5`

`-qinline+procedure1[:procedure2[:…]]`

Aggressive loop analysis and transformations

`-qhot=level=[0-2]`

Loop unrolling

`-qunroll`

Intra-/inter-procedural optimization (compiling and linking)

`-qipa`

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Tuning Runtime Environment

Network

- Topology on BG/Q: 5D Torus
  A×B×C×D×E(×T)

Shape

- Extension of a partition in A, B, C, and D direction in terms of midplanes

Mapping

- Assignment of processes to nodes and cores
- Best performance for nearest-neighbor communication
- Processes should be mapped accordingly
  - *Optimal mapping depends on application / communication pattern*
  - *Might be performance critical for jobs sizes > 1 midplane*

Choosing Shape and Mapping

Shape

```
#@bg_shape  = <AxBxCxD> #JUQUEEN: 2x7x2x2 maximum
#@bg_rotate = False|True
```

Mapping

1. Specified as a permutation of ABCDET (rightmost fastest)
2. Specified via a map file

```
1. runjob --mapping ACBDET
2. Runjob --mapping <mapfile>
```

- Default mapping: ABCDET
- Good for 1D communication patterns (communication with task ± 1)

Guidance and Map File

Example for Mapping considerations:

Job size of 1 midplane with 16 tasks/node

Default mapping: ABCDET = 4x4x4x4x2x16

→ Good for simulations with a 2D decomposition 256×32 or 64x128

→ For simulations with a 2D decomposition 128x64 chose TEDCBA

A map file is a plane ASCII file

The n^{th} line contains the coordinate of the n^{th} task

```
0 0 0 0 0 0 # task 0; coordinates ( 0, 0, 0, 0, 0, 0)
1 0 0 0 0 0 # task 1; coordinates ( 1, 0, 0, 0, 0, 0)
2 0 0 0 0 0 # task 2; coordinates ( 2, 0, 0, 0, 0, 0)
[...]
```

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

MPI Tuning – BG/Q Extensions

Blue Gene/Q specific MPI extensions (MPIX)

- Only C/C++ interfaces available, Fotran interface requested
- Include header: **`#include <mpix.h>`**

Examples

```
int MPIX_Torus_ndims(int *numdim)
```

Determines the number of physical hardware dimensions

```
int MPIX_Rank2torus(int rank, int *coords)
```

Returns the physical coordinates of an MPI rank

```
int MPIX_Torus2rank(int *coords, int *rank)
```

Returns the MPI rank with the physical coordinates specified

```
int MPIX_Hardware(MPIX_Hardware_t *hw)
```

Returns information about the hardware the application is running on

Example: MPIX_Hardware(MPIX_Hardware_t *hw)

C/C++

```
typedef struct
{
  unsigned prank;                          // Physical rank of node
  unsigned psize;                          // Size of partition
  unsigned ppn;                            // Processes per node
  unsigned coreID;                         // Process ID
  unsigned clockMHz;                       // Frequency in MHz
  unsigned memSize;                        // Memory in MB
  unsigned torus_dimension;                // Actual torus dimension
  unsigned Size[MPIX_TORUS_MAX_DIMS];      // Max. torus dimensions
  unsigned Coords[MPIX_TORUS_MAX_DIMS];    // Node's coordinated
  unsigned isTorus[MPIX_TORUS_MAX_DIMS]; // Wrap-around dims?
  unsigned rankInPset;
  unsigned sizeOfPset;
  unsigned idOfPset;
} MPIX_Hardware_t;
```

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

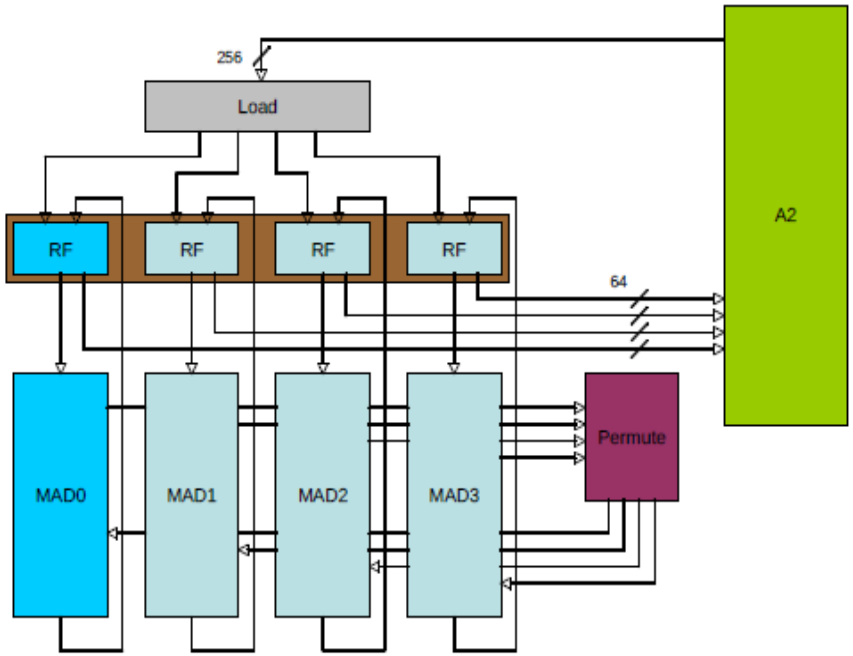
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Quad Floating Point Extension Unit (QPX)

4 double precision pipelines, usable as:

- scalar FPU
- 4-wide FPU SIMD (Single Instruction Multiple Data)
- 2-wide complex arithmetic SIMD

8 concurrent floating point ops (FMA) + load + store



IBM XL Compiler Support for QPX

Usage of QPX

- Compiler flag **`-qsimd=auto`**
- ***Check*** that simd vectorization is actually done!
  - **`-qreport`**
  - **`-qlist`**

```
>>>> LOOP TRANSFORMATION SECTION <<<<
[...]
-----------------------------------
0 9 1     Loop with nest-level 1 and
          iteration count 1000 was
          SIMD vectorized
[...]
```

```
>>>> LOOP TRANSFORMATION SECTION <<<<
[...]
-----------------------------------
0 9 1     Loop was not SIMD vectorized
          because the loop is not the
          innermost loop.
0 10 1    Loop was not SIMD vectorized
          because it contains memory
          references with non-
          vectorizable alignment.
```

QPX Usage – Hints for the Compiler

Compiler needs hints

- Hint compiler to likely iteration counts
- Instruct compiler to align fields
- Tell that FORTRAN assumed-shape arrays are contiguous
  -qassert=contig

Fortran

```
real*8 :: x(:),y(:),a
!ibm* align(32, x, y)
!ibm* assert(itercnt(100))
do i=m, n
  z(i) = x(i) + a*y(i)
enddo
```

C/C++

```
double __align(32) *x, *y;
double a;
#pragma disjoint(*x, *y)
#pragma disjoint(*x, a)
#pragma ibm iterations(100)
for (int i=m;i<n;i++)
  z[i] = x[i] + a*y[i]
void foo(double* restrict a1,
         double* restrict a2) {
for (int i=0; i<n; i++) a1[i]=a2[i];
}
```

QPX Example using Compiler Intrinsics

```
…
typedef vector4double qv;
qv dx,dy,dz,dx2,dy2,dz2
for (i=0;i<4;i++)
{
…
  xd[i] = xdipl[j];
  yd[i] = ydipl[j];
  zd[i] = zdipl[j];
}
dx2 = vec_mul(dx,dx);
dy2 = vec_mul(dy,dy);
dz2 = vec_mul(dz,dz);

d = vec_swsqrt(dx2+dy2+dz2);
…
```

→Talk:
Stefan Krieg, JSC
Vectorisation using QPX instrinsics
5. February 11:00

Source: IBM Corporation

Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory

Thread Level Speculation (TLS)

Parallelize potentially dependent serial fragments

- runtime creates threads for each speculative section
- threads run parallel and commit *in order* if no conflict
- on conflict, all threads except current master is rolled back

Performance governed by tradeoff of overhead and conflict probability

Number of times to try rollback before non-speculative execution can be set

Hardware Limitation: maximum of 16 domains

Thread Level Speculation

Enabling of TLS by compiler flag and pragmas

```
-qsmp=speculative
```

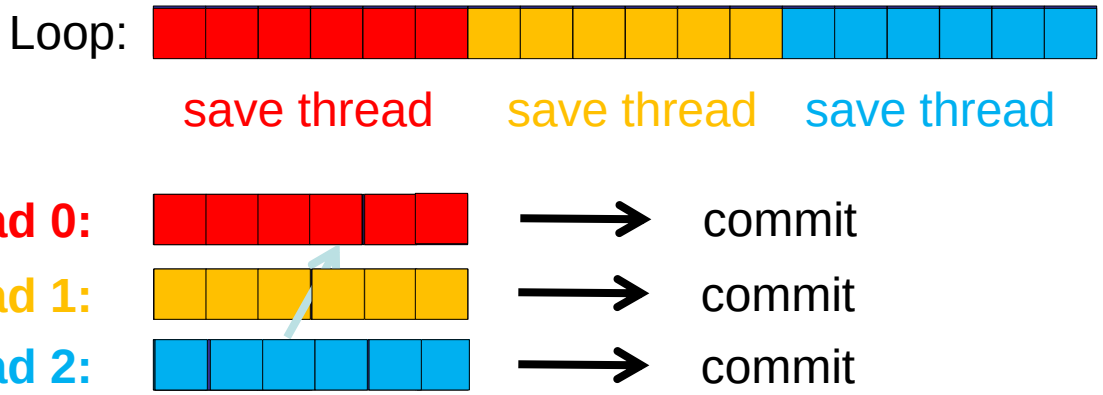
Fortran

```
!SEP$ SPECULATIVE DO
do i = 1, N
  call code_to_be_spec(i)
enddo
!SEP$ END SPECULATIVE DO
!SEP$ SPECULATIVE SECTIONS
call some_code()
!SEP$ SPECULATIVE SECTION
call other_code()
!SEP$ END SPECULATIVE SECTIONS
```

C/C++

```
#pragma speculative for
for (int i=0;i<N;i++) {
code_to_be_spec(i);
}
#pragma speculative sections
{
#pragma speculative section
{ some_code(); }
#pragma speculative section
{ other_code(); }
}
```

Thread Level Speculation




```
export SE_MAX_NUM_ROLLBACK=N
```

→Talk:
Thilo Maurer, IBM
Memory hierachy, transactional memory, speculative execution
5. February 09:30

Transactional Memory

- Mechanism to enable atomic operations on arbitrary set of memory locations
- Application needs to allow that transactions commit *out-of order*
- May be used to parallelize workload into collaborative but independent tasks on shared data
- Hardware detects write/read conflicts
- Runtime rolls back on failure

Transactional Memory

→Talk:
Thilo Maurer, IBM
Memory hierachy, transactional memory, speculative execution
5. February 09:30

Enabling by compiler flag and pragmas

```
-qtm
```

Identification of atomic code blocks:

Fortran

```
!$omp parallel
!$omp do private(i)
do i = 1, N
!TM$ TM_ATOMIC SAFE_MODE
  call code_to_be_atomic(i)
!TM$ END TM_ATOMIC
enddo
!$omp end do
!$omp end parallel
```

C/C++

```
#pragma omp parallel
{
#pragma omp for
for (int i=0;i<N;i++) {
#pragma TM_ATOMIC SAFE_MODE
    {
      code_to_be_atomic(i);
    }
  }
}
```

```
export TM_MAX_NUM_ROLLBACK=N
export TM_REPORT_...=...
```

User Information and Support

→Talk:
Paul Gibbon, JSC
Support structure at JSC
4. February 12:00

Information about JUQUEEN

- JSC websites at
  http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html

Dispatch and User Support

- Applications for accounts (for approved projects)
  Forschunszentrum Jülich GmbH,JSC, Dispatch, 52425 Jülich
  Tel: +49 2461 61 5642, Fax: +49 2461 61 2810
  email: dispatch.jsc@fz-juelich.de

- User Support
  Tel: +49 2461 61 2828
  email: sc@fz-juelich.de