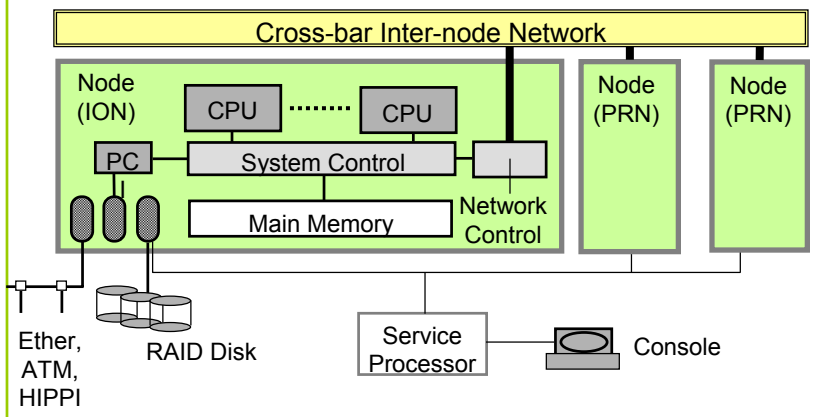
Hitachi SR8000
Programming Models and Tuning

Rolf Rabenseifner
rabenseifner@hlrs.de

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

System Architecture SR 8000



Basic Programming Models

- Basic programming models on the compute partitions
  - MPI: message passing parallelization
    - **between nodes**
    - **inside of a node**
  - OpenMP: thread-based shared memory parallelization
    - **based on directives**
    - **inside of a node**
  - Compas: automatic thread-based shared memory parallelization
- Scalar programming for the command partition (ION)

Hybrid Programming Models

- http://www.hlrs.de/organization/par/services/models/models_sr8k.html
- multi-node:
  - MPI & Compas
  - MPI & OpenMP
  - MPI-MPP (massively parallel processing)
- single node – parallel:
  - Compas
  - OpenMP
  - MPI
- single compute processor – scalar
- single command processor – scalar

MPI & Compas

MPI + Compas	Each 8-CPU SMP node is used as one MPI process. Inside of each MPI process, pseudo-vectorization (Compas) with the Fortran or C compiler is used.	
	Compile	**mpif90** -OSS -parallel
		mpicc -O4 -pvec +Op -parallel
	Batch	qsub -q multi -N #nodes *jobscript* ***with:*** **mpiexec** -N $NODES -n $NODES *my_appl my_options*
	Interact.	**mpiexec** -p multi -N #nodes -n #nodes *my_appl my_options*

MPI & OpenMP

MPI + OpenMP	Each 8-CPU SMP node is used as one MPI process. Inside of each MPI process, the process is parallelized into eight threads with OpenMP.	
	Compile	**mpif90** -OSS -parallel **-omp**
		mpicc -O4 -pvec +Op -parallel **-omp**
	Batch	qsub -q multi -N #nodes *jobscript* ***with:*** **mpiexec** -N $NODES -n $NODES *my_appl my_options*
	Interact.	**mpiexec** -p multi -N #nodes -n #nodes *my_appl my_options*

MPI – MPP

MPI-MPP	Massively Parallel Processing MPI: Each CPU is used for one MPI process, i.e., on each SMP node with 8 CPUs, up to 8 MPI processes can run.	
	Compile	**mpi**f90 –OSS –**no**parallel
		mpicc –O4 –pvec +Op –**no**parallel
	Batch	qsub –q multi –N #nodes *jobscript* ***with:*** **mpiexec** –N $NODES –n **$PROCESSES** *my_appl my_options*
	Interact.	**mpiexec** –p multi –N #nodes –n **#processes** *my_appl my_options*

Remarks:

- –**no**parallel = no thread-parallelism, only message passing, i.e., MPP
- If using only 1–8 MPI processes inside of one node:
  - batch: –q single –N 1
  - interactive: –p single –N 1

Compas

Compas	The application has only one process on one node and it is parallelized by using the 8 CPUs of the node via the Compas pseude-vectorization:	
	Compile	f90 –OSS –parallel
		cc –O4 –pvec +Op –parallel
	Batch	qsub –q single –N 1 *jobscript* ***with:*** **prun** *my_appl my_options*
	Interact.	**prun** –p single *my_appl my_options*

OpenMP

OpenMP	The application has only one process on one node and it is parallelized by using the 8 threads on the 8 CPUs of the node via the OpenMP compilation flag.	
	Compile	f90 -OSS -parallel **-omp**
		cc -O4 -pvec +Op -parallel **-omp**
	Batch	qsub -q single -N 1 *jobscript* ***with:*** **prun** *my_appl my_options*
	Interact.	**prun** -p single *my_appl my_options*

Remarks:

- The “single” and “multi” partitions are overlapping.

Scalar

Scalar	The application has only one process with a single thread on one CPU of a node.	
	Compile	f90 -OSS -**no**parallel
		cc -O4 -pvec +Op -**no**parallel
	Batch	qsub -q scalar -N #nodes *jobscript* ***with:*** **hrexec** *my_appl my_options*
	Interact.	**hrexec** -p scalar *my_appl my_options*

Remarks:

- In the scalar queue and partition, the processes can be moved to the 9th CPU in each node.
- “Scalar” and “single” are using the same nodes.

ION = Command partition

ION	The default model used for all commands, like ls, vi, ...	
	Compile	f90 -OSS -**no**parallel
		cc -O4 -pvec +Op -**no**parallel
	Execute	**just start it:** *my_appl my_options*

Remarks:

- On nearly any node and CPU.
- The processes can be moved to the 9th CPU in each node when an MPI, Compas or OpenMP job is started there.

General Remarks I.

- **mpirun**, **mpiexec**, and **prun** are setting the environment variable JOBTYPE=E8S. This is a local modification **at HLRS**!
- The partition can be chosen by setting the environment variable DEFPART (as in the batch-job example) or with the option –p (as in the interactive example).
- In batch-jobs, one **must** use the partition-name automatically stored in the environment variable $QSUB_PARTNAME.
- In the **multi** queue and partition, the processors are dedicated to the application. Therefore this queue and partition should be used only for parallel execution on more than one node.
- In the **single** queue and partition, the processes are gang-scheduled with a time-slice of 2 seconds. In the current version of the operating system, gang-scheduled is only available for single-node execution.

General Remarks II.

- mpiexec and mpirun are using the PATH environment variable to find the executable my_appl.
- –OSS / –O4 enables most powerful optimization and the pseudo-vectorization.
- grep F90OPTS /usr/ccs/cfg/f90.cfg shows **additional default options used by the Fortran compiler** on your SR8000 platform, e.g.,
  - **–i,P** --> Specifies the language extended specification. See manual Optimizing FORTRAN90 User's Reference for details.
  - **–e** --> Enables compatibility with other vendors and with Fortran standard.
  - **–I/usr/include** --> Sets the include path.
- Same must be set on your cross-compiler platform, e.g., in hwwhpv:/usr/SR8000/USR/ccs/cfg/f90.cfg

Remarks on MPI-MPP

- By default the environment variable MPIR_RANK_NO_ROUND=yes is exported:
  - This is a local modification **at HLRS**!
  - This implies that the ranks in MPI_COMM_WORLD are allocated sequentially to the nodes, i.e., rank=0, 1, 2, ... are allocated on the first node, the next ranks on the next node, and so on.
  - With export MPIR_RANK_NO_ROUND=no (on sh, ksh, bash) or setenv MPIR_RANK_NO_ROUND no (on csh, tcsh), the ranks in MPI_COMM_WORLD are allocated round-robin, i.e., rank=0 on node 0, rank=1 on node 1, ... and then again rank=#nodes on node 0, rank=#nodes+1 on node 1, ...
  - This default is exported by /usr/local/rc/profile, which should be called by your .profile .
- -OSS / -O4 **-noparallel** enables most powerful optimization, but **inhibits the automatic Compas SMP-parallelization**.

Batchjob-Script Recommendation on Hitachi SR8000

```
#!/bin/ksh
#@$-eo            # std-error into file std-out
#@$-N  2          # number of nodes
#@$-lE  3:20:00   # set a max. per-request etime
                  # limit of 3h 20min
#@$-lT  6:40:00   # set a max. per-request cpu
                  # limit of 6h 40min
#@$-lM  4000mb    # memory limit per request:
                  # maximum: 6500MBytes in "multi"
                  # in single recommended: < 1500 MB
#@$-ls  10mb      # per-process stack-segment size limits
#@$-q multi       # Queue request to pipe queue: multi
                  # Available Pipe Queues:
                  #   multi : for multi-nodes jobs
                  #   single: for single-node parallel jobs
                  #   scalar: for serial jobs (only one CPU)
#
#@$               # no more NQS parameters
. /etc/profile    # setup shell environment
. ~/.profile      # user specific
DEFPART="$QSUB_PARTNAME"  # set partition for ...
export DEFPART            # ... parallel jobs
```

continued

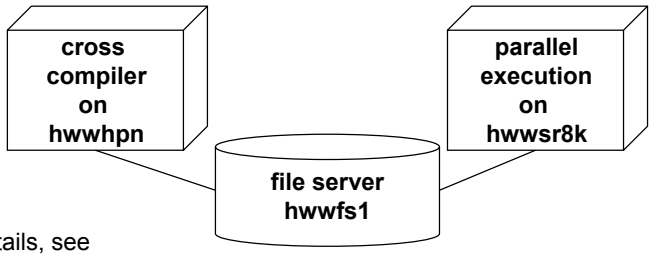
```
NODES=1                          # set number of Nodes
if [ "$QSUB_NODE" -gt 0 ]; then
  NODES="$QSUB_NODE"
fi
export NODES
(( PROCESSES=$NODES * 1 ))       # set number of processes
                                 # =nodes for MPI-COMPAS jobs
                                 #       and for MPI-OpenMP jobs
# or
#(( PROCESSES=$NODES * 8 ))      # set number of processes
                                 # =nodes*8cpu's  for MPI jobs
export PROCESSES
cd $SCRDIR/work_dir              # go into working directory
date
echo  start application on Nodes: $NODES in partition: $DEFPART
                                 # start application using all nodes
                                 # (mpirun doesn't search $PATH)
mpiexec -N $NODES -n $PROCESSES $SCRDIR/work_dir/my_prog \
                                                  my_options
# or old-fashion mpirun:
#mpirun -n $NODES -np $PROCESSES $SCRDIR/work_dir/my_prog \
                                                  my_options
date
```


Other Programming Models

- HPF (pf90)
- further information on
  - www.hlrs.de/organization/par/services/models/hpf/
  - www.hlrs.de/organization/par/services/tools/compiler/pghpf.html

Cross Compilation

- All cross compilers and development tools (C, C++, KCC, KDB, Fortran90/95, include, lib) for Hitachi SR8000 are installed on the HP V2250 (hwwhpv) and HP N4000 (hwwhpn).
- At the moment only users of HLRS projects have access to the platform hwwhpv and hwwhpn.
- Cross compilation is significantly faster than native compilation



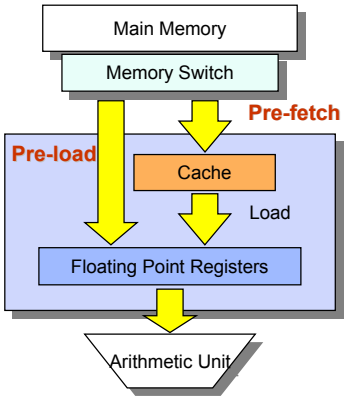
- Details, see http://www.hlrs.de/hw-access/platforms/sr8k/crosscompiler.html

Tuning

- First step: Tuning the efficient usage of each processor!
  - Recapitulating the SR8000 vector features
  - Implications for the applications
  - Performance profiling and tuning
- Second step: Tuning the parallelization
  - e.g. with VAMPIR (already discussed)

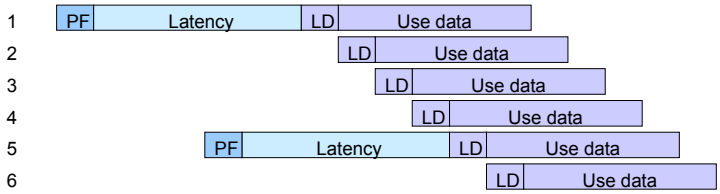
CPU Architecture

- 16 bytes/cycle memory BW
- 128 Kbyte L1 cache
- 160 FP registers
- 2 FP pipelines
- 4 flops/cycle
- Pre-fetch:
  - 16 open transactions
  - 16 byte/cycle
  - loads total cache line
- Pre-load:
  - bypassing the cache
  - only 8 byte/cycle
  - allows any random access
- Based on IBM Power PC (CPU), vector-registers and -features included by Hitachi



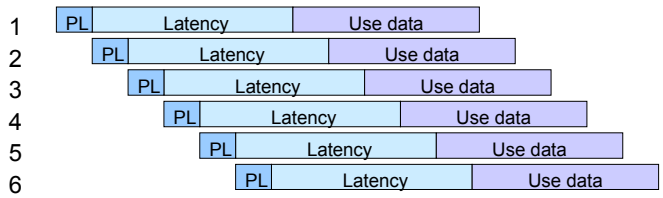
Pre-fetch

Iteration



- Pre-fetch 128 bytes to cache
- Follow by LD to register

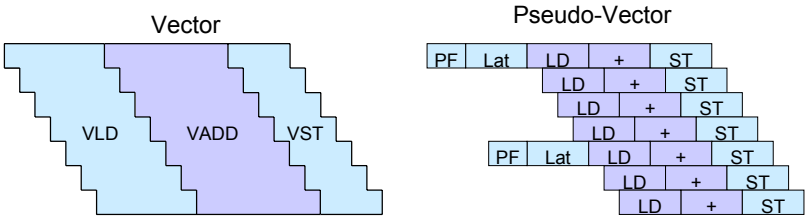
Pre-load



- Pre-load 8 bytes to register
- LD not required

Pseudo-vector Processing

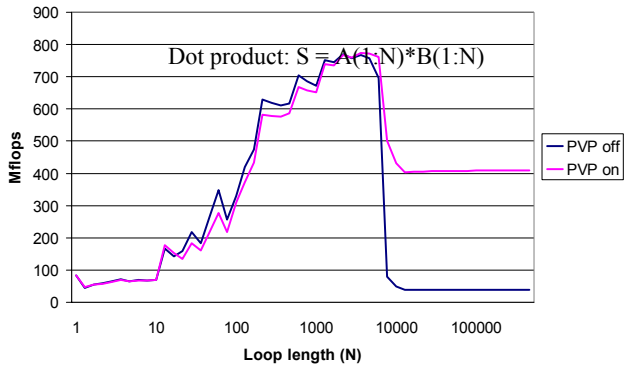
A(:) = A(:) + N



Pseudo-vectorization has nothing to do with parallelization!

It is just the way, the SR8000 is vectorizing your code!

Effect of PVP



Implications

- Make use of predicates, inlining, speculative prefetches, and vectorized math routines.
- Make use of loop unrolling.
  The SR8000 is both, vector and cache architecture.
- Indirect accesses:
  - The method of choice strongly depends on whether the code can be cache blocked.
  - Use pre-load if the memory access is in a completely arbitrary fashion without any re-use of data
  - Note, pre-loads have only half of the memory bandwidth through the cache.
  - Therefore you won't see more than 2 Gflop/s per node.

Source: Christian Simmendinger: SR8000 – Pseudo-Vectorization. March 2001.

Profiling

- Profiling flags: –Xmonitor –Wb,Phopt,monitor_loop=FALSE (C)
  –Xparmonitor (Fortran)
- Compiler-log: –loglist
- Linking with: -lpl
- Compiler listing: on *sourcefile*.log
- Profiling output at runtime: on pl_*<process_id>*.txt
  in the current working directory

These options are always helpful when developing the code!

Source: Christian Simmendinger: SR8000 – Performance Profiling and Tuning. March 2001.

Tuning

- Pseudo-vectorization:
  - critical log entries:
    - **prefetch not applied**
    - **preload not applied**
    - **SWPL not applied (SWPL = software pipeline optimization)**
    - **PVP not applied (PVP = pseudo vector processing)**
  - check the code and help the compiler:
    - ***soption unroll(n)**
    - ***soption predicate**
    - ***voption indep(a)**
    - ***voption prefetch(a)**
    - ***voption preload(a)**
    - ***voption nopreload(a)**
    - ***voption speculative**

Source: Christian Simmendinger: SR8000 – Performance Profiling and Tuning. March 2001.

Support at HLRS

Assistance on tuning of your

- vector code:
  - Department **Numerical Methods & Libraries**
    www.hlrs.de/organization/num/
- parallelization:
  - Department **Parallel Computing**
    www.hlrs.de/organization/par/

Summary

- The Hitachi SR8000 supports all relevant programming models.
- The different partitions allow the efficient usage of the total system with a mix of
  - multi-node message-passing parallel (vector-code) applications,
  - single node shared memory parallel (vector-code) applications,
  - single processor vector-code applications.
- The pseudo-vectorization (pvp) is done by the compiler.
- The compiler may need help:
  - In the tuning process, directives may assist the vectorization.