Pseudovectorization, SMP, and Message Passing on the Hitachi SR8000-F1

Matthias Brehm, Reinhold Bader, Helmut Heller, and Ralf Ebner

Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften,
Barer Straße 21, 80333 München,
Germany
{Brehm, Bader, Heller, Ebner}@lrz.de
http://www.lrz-muenchen.de/services/compute/hlrb

Abstract. The Leibniz-Rechenzentrum in Munich has started operating a 112-node Hitachi SR8000-F1 with a peak performance of 1.3 Teraflops in the second quarter of 2000, the fastest computer in Europe. In order to make use of the full memory bandwidth and hence to obtain a significant fraction of the peak performance for memory intensive applications, the compilers offer preload and prefetch optimization strategies to pipeline load/store operations, as well as automatic parallelization across the 8 processors contained in every node. The nodes are connected by a conflict-free crossbar, enabling efficient communication via standard message-passing interfaces. An overview of the innovative architectural concepts is given. We demonstrate to which extent the capabilities of the compiler to automatically pseudovectorize/parallelize typical application code are sufficient to produce well-performing code.

1 Aiming for Top Level Computing

In the first quarter of 2000, the Leibniz-Rechenzentrum (LRZ) Munich installed a Hitachi SR8000-F1 intended to serve as Top-Level Compute Server in Bavaria (the German acronym HLRB will be used in the following); this machine will be again enlarged by approximately half its present computing power in a second installation phase in 2Q2002. In installation Phase I the system consists of 112 nodes. Each pseudo-vector node contains 9 CPUs, 8 of which are available for computational tasks, and 8 Gigabytes memory which is accessible from the processors in a shared-memory model. The CPUs are similar to the IBM Power Architecture, with proprietary extensions added by Hitachi (cf. Section 2.1). Since the processors are operated at a frequency of 375 MHz and 4 floating point operations can be executed per cycle, each pseudo-vector node yields 12 GFlops peak performance. Thus the HLRB has a peak performance of 1.344 TFlops. The ninth processor on each node is needed as a service processor. The nodes of the SR8000-F1 are inter-connected via a three-dimensional crossbar with a bi-directional bandwidth of 2x950 MB/s between two nodes and a hardware latency of about 5 microseconds. Further details of the HLRB hardware are given in Tables 1 and 2 below.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 1351–1361, 2000.

The LINPACK performance value of the HLRB is 1035 GFlops and a sustained application performance has been measured to 450 GFlops. Hence, LRZ is currently operating the fastest computer within Europe. Usage of the HLRB will be open to German research projects which need high sustained performance and are presently not feasible on any other existing computing platform. Resources will be allocated to individual projects after a peer review process. Vectorizable Codes will be preferred, however the SR8000 architecture is sufficiently flexible that the system may be used in MPP mode as well.

2 The Innovative Architecture of the SR8000-F1

The architecture of the SR8000-F1 allows the usage of the vector programming paradigm and the scalar SMP-Cluster programming paradigm on the same machine. This is achieved by combining the superscalar RISC CPUs into a virtual vector CPU. In a traditional vector CPU the vectorized operations are executed by a vector pipe which delivers one or more memory references per cycle to the CPU. On the Hitachi SR8000-F1 the vectorizable operations are distributed among the 8 effectively usable CPUs of a node ("COMPAS", COoperative Micro-Processors in single Address Space); furthermore in case of memory-bound computing, specific memory references can be loaded into the registers or the caches some time ahead of actual use ("PVP", Pseudo-Vector-Processing). These two properties of the SR8000-F1 nodes especially contribute to the high efficiency obtained in comparison to other RISC systems.

	Phase I Configuration 1Q2000	Phase II Configuration 2Q2002
Number of SMP-Nodes	112	168
CPUs per Node	8 (+1 Service)	8 (+ 1 Service)
Number of Processors	112*8 = 896	168*8 = 1344
Peak Performance per CPU	1.5 GFlop/s	1.5 GFlop/s
Peak Performance per Node	12 GFlop/s	12 GFlop/s
Peak Performance SR8000	1344 GFlop/s	2016 GFlop/s
LINPACK Performance of the whole System	1035 GFlop/s	to be measured
Performance from main memory (most unfavourable case)	163.5 GFlop/s	244 GFlop/s
Memory per Node	8 GByte	8 GByte
Memory of total system	928 GBytes	1344 GBytes
Aggregated Disk Storage	7.4 TBytes	10 TBytes
Bidirectional Communication bandwidth using MPI	2x950 MByte/s (Hardware: 1 GByte/s)	2x950 MByte/s (Hardware: 1 GByte/s)

Table 1. Hardware Overview of LRZ's HLRB.

Processor and Memory Characteristics	
Frequency and Processor Cycle	375 MHz (2.67 nanoseconds)
Maximum Number of Operations per Cycle	4
Number of Floating Point Registers	160 (Global: 32, Slide: 128)
Number of Integer Registers	32
Data Cache Size	128 KB (write through, 4-way set associative, direct mapped)
DCache line size	128 Bytes
DCache bandwidth to registers	32 Bytes / cycle
Memory Frequency and mem-cycle	250 MHz (4 ns)
Maximum Number of Loads from Memory	16 Bytes / mem-cycle
Bandwidth to Memory per Processor	4 GB/s (32 GB/s peak for 8-processor node)

Table 2. Properties of the processor and memory system used by Hitachi in the Phase I installation. *It has not yet been decided whether a more advanced CPU model will be used in the Phase II upgrade.*

2.1 Pseudo-Vector-Processing (PVP)

Hitachi's extensions to the IBM POWER instruction set, which improve the memory bandwidth, alleviate the memory bottleneck, which is the main deficit of RISC-based High Performance Computing. This property, called Pseudo-Vector Processing by Hitachi, may be used by the compiler to obtain data either directly from memory via preload or via prefetch through the cache, depending on how the memory references are organized (see Fig. 1 below). The concept of PVP may be illustrated by the following example loop:

```
DO I=1,N
   A(I) = B(I) + C(I)
ENDDO
```

Using prefetch operations, which may be overlapped with the floating point operations, one obtains the sequence shown in the right part of Figure 2. Prefetch is not very efficient when the main memory is accessed non-contiguously because the prefetched cache line may contain unnecessary data. To improve this situation the preload mechanism was implemented. Preload transfers element data directly to the registers as illustrated in Figure 3. The physical registers are mapped to logical addresses via a sliding window technique. A special instruction (sliding window step) is used to update the slide window base value which is held in a special purpose register.

2.2 Cooperative Micro Processors in Single Address Space (COMPAS)

COMPAS enables the automatic distribution of computational work of a loop among the 8 CPUs of an SMP node by the compiler (autoparallelization) and the accompanying hardware support for synchronization. COMPAS may also

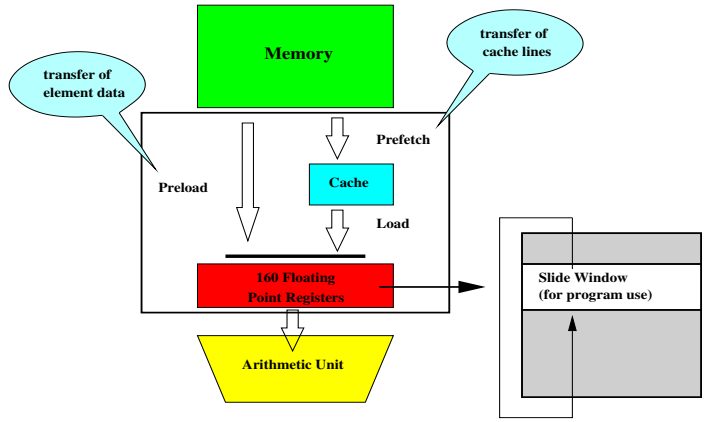

Fig. 1. Pseudo-vectorization: Prefetch and Preload.

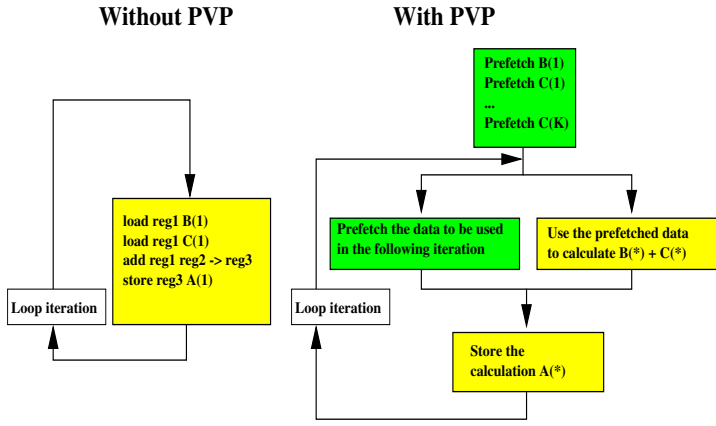

Fig. 2. Loop structure without and with PVP Prefetch.

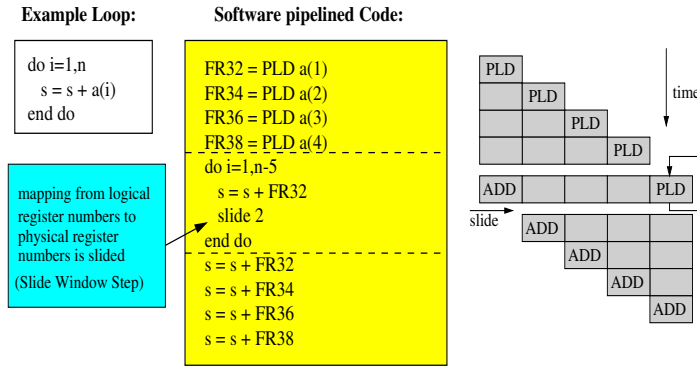

Fig. 3. Loop Structure with PVP Preload.

be utilized by codes which use the nodes as 8-way SMP-nodes via OpenMP, since version 1.0 of the OpenMP standard is implemented as part of the Hitachi Fortran Compiler.

3 Benchmark Results and Principles for Code Optimization

The most important criterion for evaluation of the offered machines was not the peak performance but the actually obtained "sustained" performance for a suite of application benchmarks. Furthermore, several additional tests were performed to obtain a measure of how the hardware performs in the least favorable situations or to evaluate scaling of MPI codes to the largest possible problem size. The examples discussed in the following subsections will also provide insights on the principles of optimizing code for the SR8000.

3.1 Memory Throughput

STREAM Benchmark This program (written by John D. McCalpin) is used to evaluate the memory bandwidth of a node. The following loops are performed:

Copy	Scale	Add	Triad
DO J=1,N C(J)=A(J) END DO	DO J=1,N B(J)=S*C(J) END DO	DO J=1,N C(J)=A(J)+B(J) END DO	DO J=1,N A(J)=B(J)+S*C(J) END DO

The vector length N used in this case was 19,121,111, corresponding to a memory usage of 437 MB for the program, and ensuring that it is really the transfer from/to memory that is measured. The memory bandwidth is plotted in Figure 4 for all types of loops indicated above. A comparison with other vendor's hardware is provided in Table 3. The machine balance is obtained as

Platform	Bandwidth (MB/s)	Peak Performance (MFlops)	Machine Balance (Flop/Word)	Remarks
SR8000-F1	22311	12000	4.3	8 Processors, 375 MHz
R12000 (SGI)	811	2400	23.7	4 Processors, 300 MHz
Pentium III	396	500	10.1	1 Processor, 500 MHz
Cray C90	103812	15360	1.2	16 Processors
NEX SX-5	583069	128000	1.8	16 Processors
IBM Nighthawk1	3872	7104	14.7	8 Processors, 222 MHz

Table 3. Overview of Triad memory bandwidth for various platforms.

the quotient of peak performance and bandwidth, the latter being expressed in 8-Byte words; lower numbers are better, as far as memory-intensive (out-of-cache) computing is concerned. Typically, RISC-based systems are at least an order of

magnitude worse than the specialized vector processors. However, Hitachi has nearly managed to bridge this gap by having a better ratio of memory cycle to processor cycle to start with, as well as being able to access memory from all processors in the SMP node simultaneously without too high losses: Of the 32 GBytes/s per node naively calculated from the single-processor bandwidth of at least 22 GByte/s can actually be obtained. In order to see how the memory bandwidth scales with the number of processors used, an OpenMP parallelized version of the STREAM benchmark was run. Figure 4 shows the efficiency

$$E(n) = \frac{\text{measured Bandwidth}(n)}{n \cdot \text{peak Bandwidth}(1)}$$

as a function of the number of threads for the various loop types. The peak bandwidth for 1 processor is assumed to be 4 GB/s (cf. Table 2). One ob-

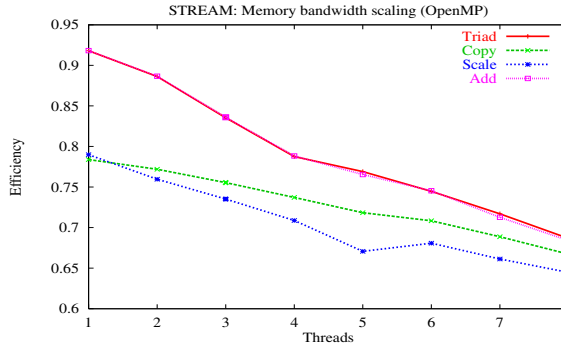

Fig. 4. Memory band-width as measured in STREAM benchmark

serves a degradation for increasing numbers of threads. The differences between Triad/Add and the other two tests are accounted for by a difference between Load and Store: Triad/Add involves 2 Loads and 1 Store, while Scale/Copy has only 1 Load and 1 Store. However it is not entirely clear why there is a measurable difference between Scale and Copy. It must be remarked that other RISC SMP machines degrade far more than the SR8000 node: On 8-way systems one can expect at most 40% of peak bandwidth. In the top ten list kept at `http://www.cs.virginia.edu/stream/top10/Bandwidth.html` the SR8000-F1 (as well as its predecessor) would be ranked at position 7.

Parallelization and Pseudo-vectorization for Triads Performing Triads for variable vector length yields information not only about the memory throughput, but of the complete register/cache/memory system. Furthermore, some of the compiler's capabilities to automatically parallelize and pseudovectorize code are investigated. Figure 5 shows the per-formance of the (3 load + 1 store, 2 operation) triad

```
DO I=1,N
   A(I) = B(I)*C(I) + D(I)
END DO
```

as a function of vector length for the four execution modes possible on the SR8000: 1) COMPAS-parallel and pseudo-vectorized, 2) COMPAS-parallel without pseudo-vectorization, 3) Non-parallel, but pseudo-vectorized, 4) Neither parallel nor pseudo-vectorized. For these measurements, the desired effect was obtained within a single program unit by inserting appropriate compiler directives. Looking at case 3) first, one observes uniformly high performance of up to 490 MFlops until the Level-1 Cache is exhausted at approximately $n = 4000$. After that, performance is governed by the memory throughput, where PVP achieves around 230 MFlops in the $n \to \infty$ limit. Using 8 IPs in parallel allows for eightfold cache usage, hence the vector-like range ends at n=30000, where a performance of 3360 MFlops is reached, the 6.86-fold of what is obtained on a single IP. The vector-like characteristic of the performance for small n is due to the domination of COMPAS startup times for short vector lengths. The size of the cache performance window strongly depends on the particular loop kernel. Very fat loop kernels may lead to register spill and hence may require to be split up, too thin kernels are fused or suitably unrolled. All of this is automatically performed by the compiler.

The $n \to \infty$ performance achieved here is 1410 MFlops, which is the 6.13-fold value of a single IP. *Hence, due to PVP the triad achieves more than 40% of its maximum performance even when leaving the cache performance window, while without PVP one obtains the RISC-typical value of around 7 %.* This is of very high importance for memory intensive computing tasks in technics and science.

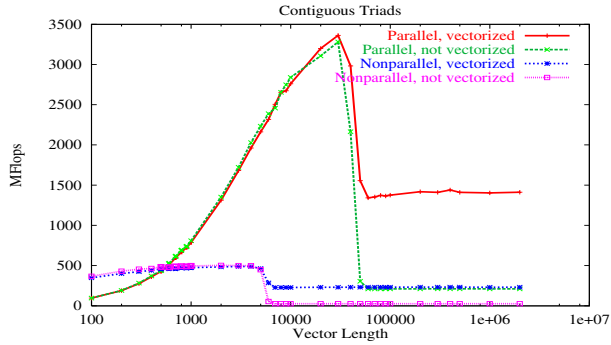

Fig. 5. Performance of Contiguous Triads as function of vector length.

3.2 Scalability of MPI Programs

Part of the LRZ benchmark suite was concerned with scalability studies. The programs

- FT (Fast Fourier Transform from NAS Parallel Benchmarks)
- MG (basic Multigrid Algorithm from NAS Parallel Benchmarks)
- HRD (ScaLAPACK call for Hessenberg transformation)

were performed on the SR8000-F1, the results of which are presented in the following. The Class C 512x512x512 three dimensional Fast Fourier Transform and the Class C Multigrid Test were performed in COMPAS mode on an increasing number of nodes. The HRD Test was performed with a fixed number of nodes but varying matrix sizes. The tests generally showed high scalability and very good performance. From the programmers point of view, it is advantageous that one has to deal only with a relatively small number of nodes instead of the eightfold higher number of processors.

Benchmark	Nodes	Processors	Performance (GFlops)	MPI Efficiency (relative to 4 nodes)
FT	4	32	15.5	1.00
	8	64	29.5	0.95
	16	128	56.9	0.92
	32	256	109.8	0.89
MG	4	32	14.8	1.00
	8	64	27.7	0.93
	16	128	46.9	0.79
	32	256	72.3	0.61

Benchmark	Nodes	Processors	Performance	Matrix Size
HRD	32	256	11.8	1000 x 1000
	32	256	52.8	10000 x 10000
	32	256	134.0	30000 x 30000

3.3 Case Studies for the Hybrid (COMPAS/OpenMP + MPI) Programming Paradigm

Parallel Vector Times Matrix As an example for the hybrid programming model, we are going to look at a simple implementation of vector-matrix multiplication. Using MPI for the distributed memory version, there are only few changes from the serial version. The structure of the code is shown in Figure 6 (for the pure MPI version the OpenMP directives are just ignored). The performance for matrix size n=10000 and n=40000 is given in Figure 7. It is obvious that the hybrid version performs much better than the pure MPI version. We first thought that this was due to the reduced amount of MPI communication. However, further examination showed that not only the communication part had shorter execution time, but also the computational part. The latter effect is caused by the increased vector length in the algorithm. For an algorithm with higher ratio of MPI-communication to computation the differences will be even more marked.

Parallel Matrix Multiplication Parallelization of Matrix Multiplication via MPI uses a ringcast scheme for matrix blocks, where the block size is chosen

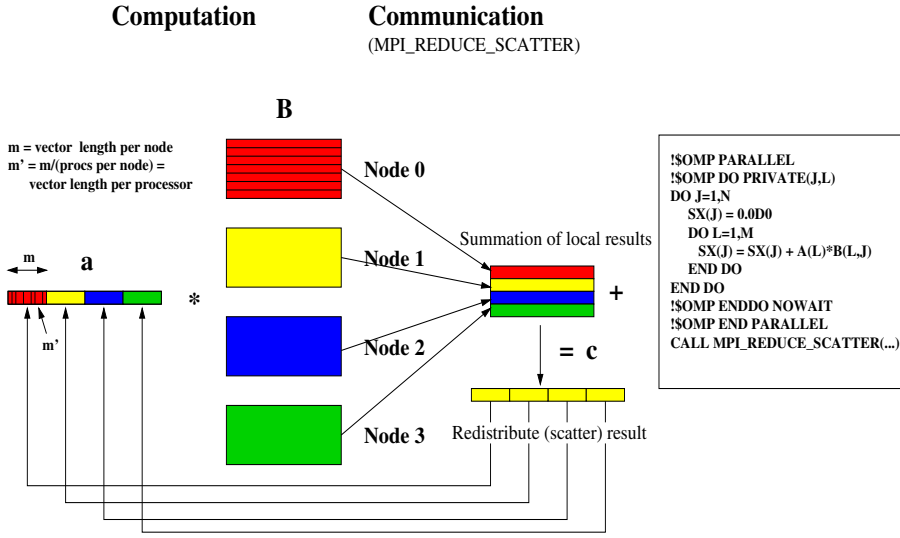

Fig. 6. Code Structure for parallel vector times matrix multiplication.

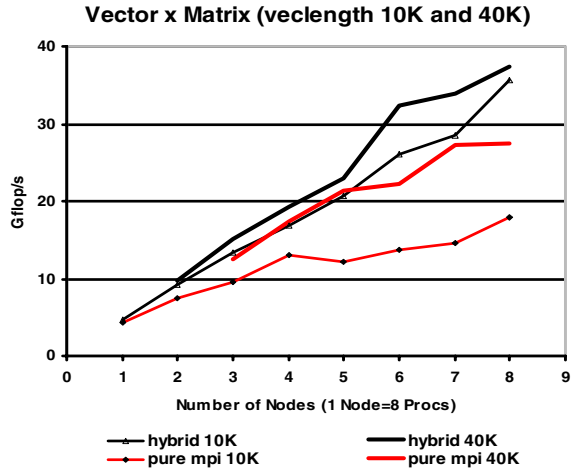

Fig. 7. Performance for vector by matrix multiplication.

optimally for a given cartesian processor grid. Given a block (IB, JB) of matrix C situated on a particular processor P, the required blocks of matrices A and B are pipelined through this processor in the course of the calculation. Between the communication steps, a normal DGEMM-call is performed. One can expect a nearly linear speed-up provided the block size is chosen large enough

to essentially circumvent MPI latency. For this test a fixed amount of memory per node was used; the matrix dimension correspondingly scales upward with the square root of the number of nodes used. Figure 8 shows the dependence of

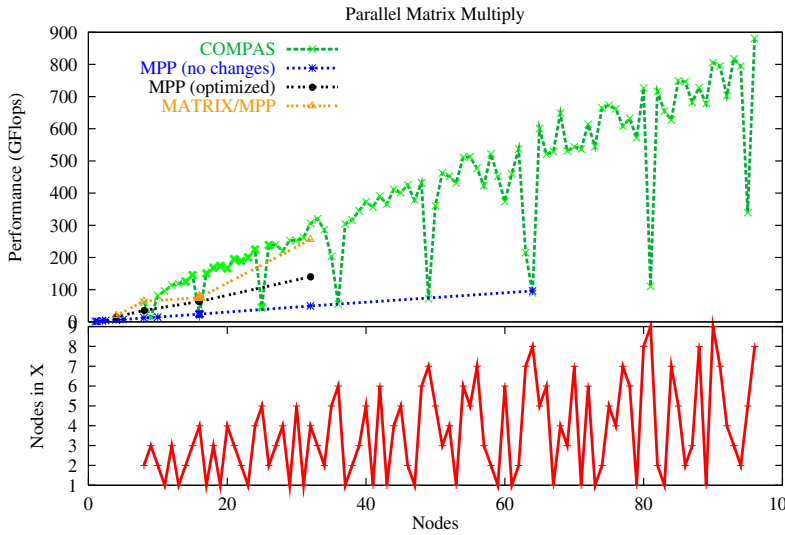

Fig. 8. Performance of parallel matrix multiplication.

performance on the number of nodes in various situations in the upper box, for the COMPAS case the lower box gives information about the node layout in the cartesian grid; e. g., for 21 nodes a 3x7 grid was used. In the COMPAS case, scaling is reasonably linear; for a yet unknown reason, square grids (4x4, 5x5, ...) appear to work particularly badly and should be avoided, at least with this particular communication pattern. Memory per node was 1.6 GByte. For the MPP (intra-node MPI) case, where 200 MByte of memory were used to obtain the same memory footprint per node as in the COMPAS case, runs with usually up to 256 processors (corresponding to 32 nodes) were performed. One obtains the very bad performance shown by the lowest curve if one simply recompiles the COMPAS code in scalar mode, using the scalar version of BLAS provided by Hitachi. The reason for this is simply that the latter library was not properly optimized at the time the tests were performed; as the "matmul_gemm" curve in Figure 9 shows, the code apparently did not optimally reuse the cache. Since the LRZ hand-coded routine ("matmul_opt2") works best for matrix sizes around 60-80, a smaller block size was chosen for a further run of the program yielding the second-lowest curve in Figure 10, which shows an improvement by a factor of 2.5. However, Hitachi also provides a proprietary library of highly optimized routines, MATRIX/MPP. Use of these – with a large block size – yields a performance comparable to COMPAS also for the internode case, as shown in the

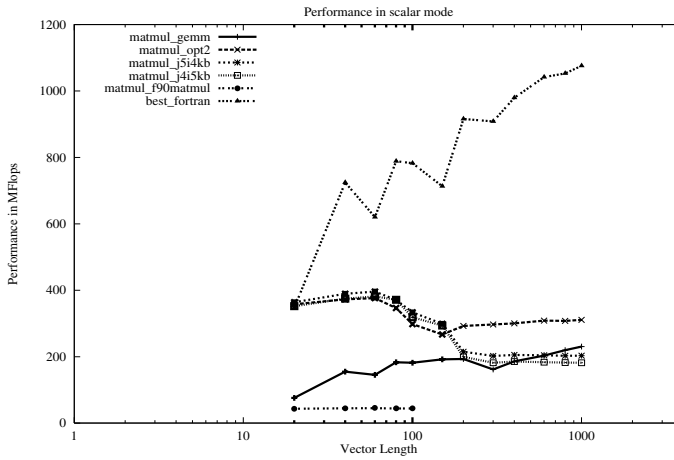

Fig. 9. Non-parallel (single-IP) matrix multiplication. The following variants are illustrated: 1. matmul_gemm: Hitachi BLAS, 2. matmul_opt2, matmul_j5i4kb, matmul_j4i5kb: blocking and loop unrolling done by hand, 3. matmul_f90matmul: Fortran 90 intrinsic, 4. best_fortran: Hitachi's MATRIX/MPP implementation.

third-lowest curve of Figure 8. The drawback of MATRIX/MPP is that there is a different API and presumably more working space has to be used.

4 Conclusion

Hitachi's SR8000-F1 installation easily manages to provide the computational power demanded by LRZ's requirements and Hitachi's own commitments. Our first tests indicate that generally more effective usage of the machine may be made by using the COMPAS mode as opposed to intra-node MPI (MPP-mode). The automatic parallelization features available with the Fortran and C compilers make it a relatively easy task to optimize high performance computing code.

5 Further Reading and Details

A Superscalar RISC Processor with 160 FPRs for Large Scale Scientific Processing:
`http://www.lrz-muenchen.de/services/compute/hlrb/system-en/Iccd99.pdf`
Node Architecture and Performance Evaluation of the Hitachi SR8000:
`http://www.lrz-muenchen.de/services/compute/hlrb/system-en/NodeArch.pdf`
An overview of the Hitachi SR8000-F1 by Hitachi may also be found at:
`http://www.hitachi-eu.com/hel/hpcc`
Some of the STREAM results were taken from:
`http://www.cs.virginia.edu/stream/top10/Bandwidth.html`