Performance Analysis of the CM-2, a Massively Parallel SIMD Computer

Jukka Helin*
helin@cc.tut.fi
Tampere University of Technology
P.O.Box 527, SF-33101 Tampere, Finland

Abstract

The performance evaluation process for a massively parallel distributed memory SIMD computer is described generally. The performance in basic computation, grid communication, and computation with grid communication is analyzed. A practical performance evaluation and analysis study is done for the Connection Machine 2 and conclusions about its performance are drawn.

1 Introduction

High-speed computing in scientific/engineering environments can be done using many kinds of computing hardware. *Massively parallel computers* have become popular in solving highly parallelizable applications. We define a massively parallel computer as *a computer whose architecture does not limit strictly the number of processors*. Currently, two computer categories fulfill this definition, namely distributed memory SIMD and MIMD computers. A massively parallel SIMD computer is a set of processors which all execute the same instruction at the same time whereas a massively parallel MIMD computer is a set of independent processors which can execute different instructions at the same time. On both computers the processors are connected together with some kind of network topology, such as tree, ring, mesh, or hypercube. SIMD computers which belong to data-parallel computers are generally considered easier to program than MIMD computers because the programmer must not take care of the synchronization between the processors. On the other hand, this limits the suitability of the computer to specific applications and can also cause inefficient use of resources. MIMD computers, or instruction-parallel computers, are considered more difficult to program than SIMD computers because the programmer has to take care of the synchronization but also being more suitable for a larger application area than SIMD computers. In Table 2 some of the current commercially available massively parallel computers are described [1].

In this paper we explain how to analyze the basic performance of massively parallel SIMD computers. We have divided the performance analysis into three parts: basic computation, basic communication, and computation with communication. Different tests developed for these parts have been applied in practice to measure and analyze the performance of the Connection Machine 2 and the tests can be easily applied to all the SIMD computers described in Table 2. The study continues the performance analysis work done on the shared memory high-speed computers [2] and on the distributed memory message passing MIMD computers [3].

2 Connection Machine 2

Connection Machine 2 (CM-2) from Thinking Machines Corp. is currently the most massively parallel computer in the market. Two models exist: CM-2 with 16K, 32K, or 64K processors and CM-2a with 4K or 8K processors. Like many massively parallel computers CM-2 is an attached processor, that is, a host computer is needed. The CM-2 can be connected e.g. to the VME bus of a Sun-4 running SunOS or to the VAXBI bus of a VAX running Ultrix. The program development software package including CM Fortran, C*, and *Lisp compilers and a Paris (PARallel Instruction Set) assembler runs on the front end. The compilers extract parallel portions of the program and generate Paris instructions from them to be broadcasted at execution time to the CM-2 through the host bus. The scalar portions of the program are executed on the front end. The "macro instructions" issued by the front-end are decoded on the CM-2 by a micro-coded sequencer which in turn broadcasts "nano instructions" to all CM-2 data processors. The CM data processors are custom designed bit-serial 1-bit ALUs packaged 16 processors per chip. Each processor has 64 or 256 Kbit of bit-addressable local memory. The processors within a chip are connected together with a special permutation network whereas the chips are connected together by a hypercube network. Three types of communication have hardware support: general routing, grid communication (also known as NEWS or North-East-West-South communication), and scanning. Actually, the scanning takes advantage of the NEWS grid communication

*The main part of this study has been done when the author worked as a visiting scientist at the Research Center Jülich GmbH (KFA), Institute for Applied Mathematics, Germany.

ICS '92-7/92/D.C., USA
© 1992 ACM 0-89791-485-6/92/0007/0045...$1.50

System	Architecture	Processor min–max	Processor type	Peak MFLOPS min–max	Network topology
Alliant Campus	MIMD	25 – 800	Intel i860	1,000 – 32,000	cluster
AMT DAP	SIMD	1,024 – 4,096	custom 8-bit	n/a	3D mesh
BBN TC2000	MIMD	8 – 512	MC88100	160 – 10,240	multistage
CM-2	SIMD	4,096 – 65,536	custom 1-bit	1,792 – 28,672[a]	hypercube
CM-200	SIMD	2,048 – 65,536	custom 1-bit	1,280 – 40,960[a]	hypercube
CM-5	MIMD	n/a – 16,000	SPARC	n/a – 2,000,000	n/a
Intel iPSC/860	MIMD	8 – 128	Intel i860	480 – 7,600	hypercube
Intel Delta[b]	MIMD	8 – 528	Intel i860	480 – 31,680	2D mesh
Intel Paragon XP/S	MIMD	66 – 4,000	Intel i860XP	5,000 – 300,000	2D mesh
Kendall Square KSR1	MIMD	8 – 1088	custom 64-bit	320 – 43,520	cluster
MasPar MP-1	SIMD	1,024 – 16,384	custom 4-bit	41 – 600	3D mesh
Meiko M40	MIMD	2 – 78	Intel i860	120 – 4,680	hypercube
nCUBE-2	MIMD	32 – 8,192	custom 64-bit	77 – 19,660	hypercube
Parsytec GC-2[c]	MIMD	64 – 256	T9000 transputer	1,600 – 6,400	3D mesh
Wavetracer DTC	SIMD	4,096 – 16,384	custom 1-bit	n/a	3D mesh

[a]32-bit computing.
[b]prototype, not commercially available.
[c]available probably late '92.

Table 1: Some commercially available massively parallel computers.

mechanism. A CM-2 *processing node* consists of two chips, or 2×16 data processors, their associated memories, hypercube interfaces and an optional 32-bit or 64-bit FPU. In Fig. 1 the architecture of the CM-2 processing node is described.

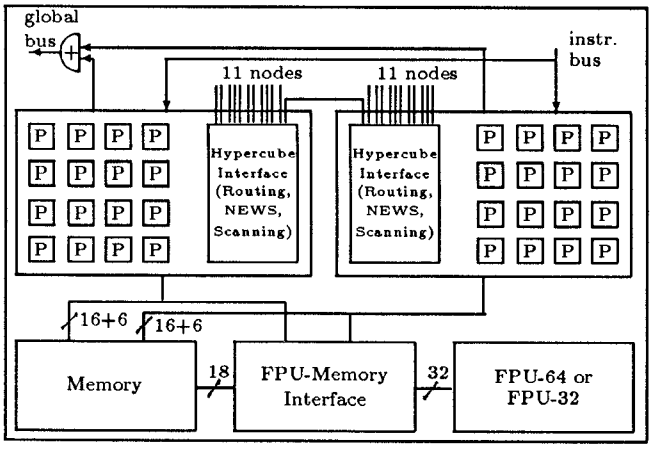


Figure 1: The architecture of the CM-2 processing node [4].

CM-2 operates at 7 MHz. Because the FPU is able to produce theoretically two 32-bit results per cycle, or two FLOP at 7 MHz, its peak performance is 14 MFLOPS. Thus, a full 64K machine having 2K FPUs has a peak performance of 28.7 GFLOPS in 32-bit arithmetic.

Since the CM-2 is a SIMD processor every data processor executes the same instruction at the same time although each processor has a flag bit which can be used to disable the processor from executing the instruction. Parallel data, that is, arrays and matrices, are allocated to *processing elements (PE)*. A PE is a data processor or a processing node depending on the two execution models (see Section 3). If there are more data items than PEs the *virtual processor (VP)* mechanism is actived and it presents the user an abstract version of the CM-2 hardware: one data item can be allocated per every VP. A program can be written assuming any appropriate number of PEs and the VPs are mapped then onto actual hardware. How many VPs are mapped onto a PE defines the *VP ratio (VPR)*. Since a complete program usually consists of parallel data of different sizes, or different VP set sizes, the VP ratio can vary during the program execution. However, only one VP set can be active at the same time. Due to VP mechanism the same program can be executed on the CM-2s having different number of PEs.

The tests were done during 1991 on three different CM-2s (see the Acknowledment) using CM operating system version 6.0 and CM Fortran compiler version 1.1. The microcode version was 6002.

In June 1991 Thinking Machines Corp. announced the CM-200 series of massively parallel computers which became available in Fall 1991. The clock speed is now 10 MHz compared to 7 MHz of the CM-2 [5]. According to TMC the sequencer has also been redesigned and the communication speed has been improved. A new 2K entry model exists. The tests we describe next to evaluate the performance of the CM-2 can be directly applied without any modifications to the CM-200.

3 Basic computation performance

Two execution models exist for executing CM-2 programs: *Paris* and *slicewise* and they can be selected on CM Fortran level by a compiler option. The slicewise model is available only on machines equipped with 64-bit FPUs whereas the Paris model is available on all the machines regardless of the FPU type. In the Paris model each 1-bit processor is considered as the basic processing element for which the data elements are allocated. On the contrary, in the slicewise model each processing node (32 data processors and a 64-

bit FPU) is regarded as the basic processing element. Under Paris all the dimensions of a VP set must be powers of two but under slicewise only the product of the dimensions of a VP set is constrained. If we denote *nproc* as the number of 1-bit data processors we have for the VP set sizes (VPSS):

$$VPSS = \begin{cases} 2^p * nproc, & p = 0, 1, 2, ... \quad \text{if Paris} \\ p * nproc/8, & p = 1, 2, ... \quad \text{if slicewise} \end{cases} \quad (1)$$

The VPSS under the slicewise model comes from the fact that the 64-bit FPUs have a vector length of 4 and thus $VPSS = p*4*nproc/32$. The performance difference of the models can be seen executing a simple parallel arithmetic operation, such as the vector addition on a CM-2 equipped with 64-bit FPUs. Sun-4 is used as the host computer in this and in all the following tests. The results are described in Fig. 2.

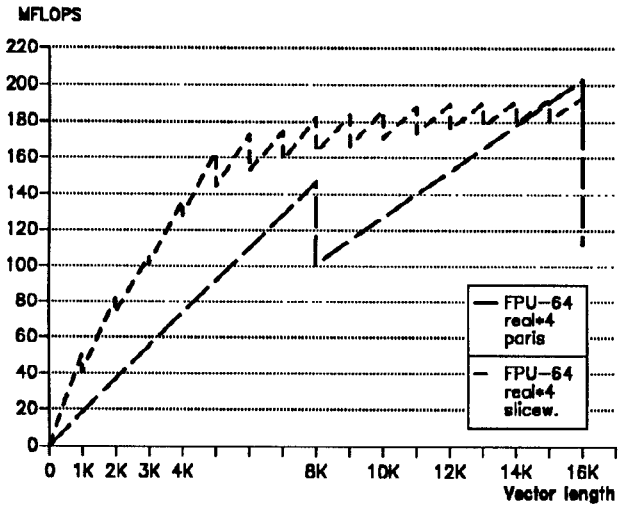


Figure 2: The performance of the $z_i = x_i + y_i$ operation on an 8K CM-2 using Paris and slicewise execution models.

The slicewise model performs good on short vectors and the performance becomes stable quite soon. On the other hand, in the Paris model it is necessary to use exact dimensioning ($2^p * nproc$) otherwise up to 50% of the performance can be lost even on long vectors. In Fig. 3 we describe the peak performance curves for the same operation using vector lengths up to 256K. The difference of 32-bit and 64-bit hardware is also demonstrated.

We can see from Fig. 3 that on the CM-2 equipped with 64-bit FPUs the highest performance, 250 MFLOPS, is achieved by the Paris execution model whereas only 205 MFLOPS is achieved by the slicewise execution model. However, the half-performance vector length $n_{1/2}$ is smaller for the slicewise execution model: about 3K compared to 7K of the Paris execution model. In 64-bit arithmetic the CM-2 equipped with 64-bit FPUs can achieve only 56% of its performance in 32-bit arithmetic. The reason for this is that even if the FPU can operate with 64-bit operands the data path between the memory and the FPU is only 32-bit wide

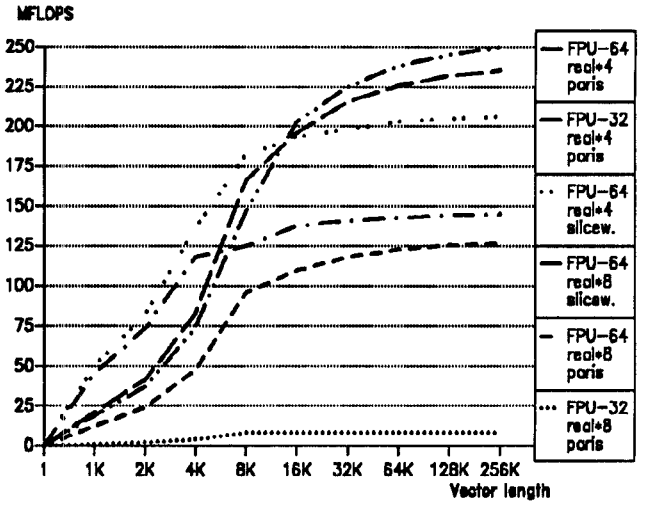


Figure 3: The performance of $z_i = x_i + y_i$ operation on 8K CM-2s.

(see Fig. 1). On the CM-2 equipped with 32-bit FPUs 64-bit arithmetic is done by the software and the performance is only 8.1 MFLOPS which is about 3% of its 32-bit performance, 235 MFLOPS.

Next we measured the maximum or asymptotic performances r_∞ in MFLOPS and half-performance vector lengths $n_{1/2}$ for a set of kernels using 8K processors of a CM-2 and one processor of an 8-processor Cray Y-MP/832 [6]. Thus, the test configurations represented 1/8 of possible full configurations on both computer systems. The calculations were performed on the CM-2 using 32-bit accuracy whereas on the Y-MP we used 64-bit accuracy because Crays do not support 32-bit floating-point data format. The FLOP weights on different vector operations have been calculated according to F. McMahon from Lawrence Livermore Laboratories: a multiply and an add are 1 FLOP, a reciprocal is 3 FLOPS, a divide and a square root 4 FLOPS, and a SIN, EXP etc. are counted as 8 FLOPS [7]. The results are given in Table 3 where we use r_{256K} as r_∞.

Generally, the Paris execution model performs better than the slicewise model on large vectors. The third column in Table 3 describes the communication mechanism (none, grid communication, or general routing) used on the CM-2 and we discuss more about them in the next section. The $n_{1/2}$ values in Table 3 are not available for the operations in which general routing is used because the router network becomes a bottleneck at higher vector lengths when more communication is needed. Usually the highest performance in these operations is achieved already at vector length 8K or at VP ratio 1. For example, at 8K the performance in the operations involving routing (the last three lines for the FPU-32 in Table 3) are 0.07, 2.33, and 11.8 MFLOPS, respectively.

Since the performance scales linearly on the CM-2 *if no communication is involved* we can calculate that the peak execution rate of the 64K CM-2 in 32-bit arithmetic is about 4.4 GFLOPS on the CM Fortran level (FPU-64, Paris exe-

Kernel	*FLOP per iter.*	*comm.*	*Computer* *CM-2, 8K processors* *FPU-64 cmf 1.1 paris*		*FPU-64 cmf 1.1 slicewise*		*FPU-32 cmf 1.1 paris*		*Cray Y-MP 1 proc 64-bit cf77 5.0*	
			$n_{1/2}$	r_∞	$n_{1/2}$	r_∞	$n_{1/2}$	r_∞	$n_{1/2}$	r_∞
$z_i = x_i + y_i$	1	none	7K	250	3K	206	6K	235	35	135
$z_i = x_i y_i$	1	none	7K	251	3K	206	6K	269	35	131
$z_i = x_i / y_i$	4	none	5K	438	1K	359	4K	232	16	169
$z_i = \sqrt{x_i}$	4	none	5K	304	1K	294	4K	173	17	57
$z_i = sin x_i$	8	none	4K	203	1K	135	4K	274	10	43
$y_i = y_i + a x_i$ (DAXPY)	2	none	8K	464	3K	401	6K	431	20	216
$x_i = a x_i$	1	none	8K	304	4K	282	6K	322	17	112
$z_i = a x_i + b$	2	none	9K	554	4K	486	6K	440	36	295
7th degree pol.	14	none	8K	434	4K	371	7K	357	0	16.1
$z_i = x_{i+1} - x_i$	1	NEWS	11K	147	9K	61.9	8K	141	40	130
IDAMAX	1	NEWS	9K	68.5	50K	454	8K	68.4	450	69.5
$s = \sqrt{\sum x_i^2}$	2	NEWS	109K	218	70K	195	46K	454	650	298
$s = \sum x_i y_i$	2	NEWS	78K	419	37K	294	45K	445	490	271
Matrix × vector	2	NEWS	5K	8.73	4K	6.76	4K	9.40	20	164
$z_i = (k_{i-250} \oplus k_{i-103})/a$	4	router	n/a	0.01	n/a	0.01	n/a	0.01	20	243
$z_i = x_{ind_i} + a$ (gather)	1	router	n/a	1.80	n/a	1.51	n/a	1.80	20	86.7
$z_{ind_i} = x_i + a$ (scatter)	1	router	n/a	4.16	n/a	3.70	n/a	4.15	21	93.2

Table 2: r_∞ and $n_{1/2}$ values for some kernels.

cution model, operation $z_i = ax_i + b$). Furthermore, if we define the *efficiency* as the actual performance divided by the theoretical performance we get for the maximum efficiency on the CM-2 only 15% (4.4/28.7). It should be noted that the maximum efficiency on a single-processor Cray Y-MP is 89% (298/333).

4 Communication performance

On distributed memory computers the communication performance can be considered as important as the computation performance. Therefore, we have paid special attention to it in this section.

The communication bandwidth of a massively parallel SIMD or MIMD computer can be measured as follows: connect all the processors using a ring topology and perform a circular shift on all processors in parallel. The operation can be done several times in order to measure the time more accurately. Fig. 4 describes the test arrangement on the CM-2.

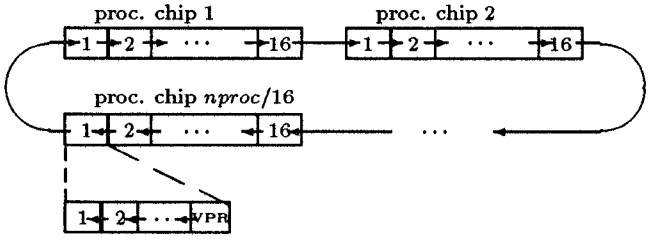


Figure 4: The measurement of the CM-2 communication bandwidth.

Since the CM-2 supports the virtual processor mechanism we have executed the test using different ring sizes. CM-2 maps consecutive ranges of virtual processors to each physical processor [8]. CM Fortran provides two intrinsic functions for shifting array elements in regular patterns along array dimensions. CSHIFT is a "wrap-around" shift which causes values that move off the edge of the array to reappear at the opposite edge (see Fig. 4). On the contrary, EOSHIFT is a "end-of" shift which discards values that move of the edge of the array and moves some specified value into the positions vacated at the opposite edge. Either the faster NEWS or the slower router communication can be used for shifting. Using the CSHIFT and Paris execution model NEWS transfers are used only if the dimensions of the argument array are powers of two and the shift distance can be broken into a few power-of-two distances. We executed the test using 4K processors of a CM-2a and ring sizes 4K, 8K, 16K, ..., 512K or VP ratios 1, 2, 4, ..., 128, respectively. The shift distance was always one. In order to measure the bandwidth with the router network the ring sizes were selected not to be powers of two. The results are shown in Fig. 5.

When EOSHIFT is used the results are the same regardless of the fact whether the ring size is a power of two or not: NEWS transfers are always used. The CSHIFT with router network can achieve only about 4.3 Kbyte/s but when the NEWS communication mechanism is used the maximum communication bandwidth per a single virtual processor increases to 180 Kbyte/s. Paris instruction **_CM_get_from_news_1L** is generated by the compiler. This will give 11.8 Gbyte/s as the total grid communication bandwidth on the 64K machine. However, since this value is reached using a high VP ratio almost all the communication is done between the virtual processors which are mapped on the same physical processor and the transfer operations can be done simply rearranging data within the local memory of a processor. Let us next study how much data is actually transferred between the physical processors.

The maximum bandwidth rate of 180 Kbyte/s is achieved

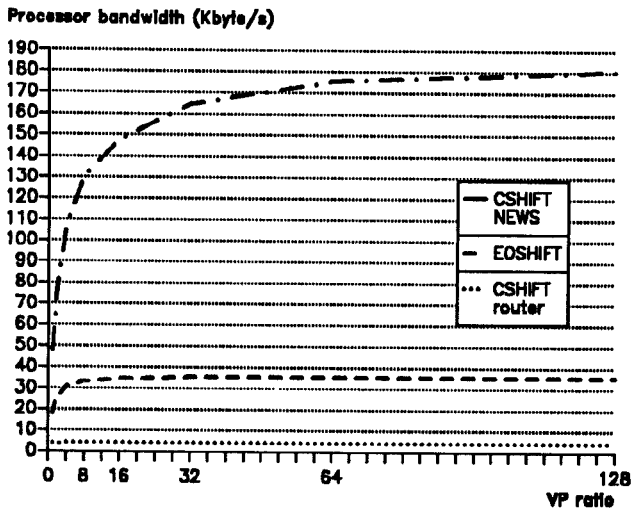


Figure 5: Processor communication bandwidth of the CM-2.

using a VP ratio of 128 which means that 128 VPs are mapped on every physical processor. This means that during our test within each group of 128 processors 127 must "send" data *serially* to another virtual processor that is within the same physical processor (see Fig. 4). Let us denote t_P as the time per 32-bit word it takes to "transfer" data between VPs belonging to the same physical processor.

As we can see from Fig. 1 there are 16 processors on each CM-2 processor chip. The processors within a chip use a special permutation circuit which can organize the processors as 1×16, 2×8, 4×4, $4\times4\times2$, $1\times2\times2\times4$, etc. grid. In our 1-dimensional ring test the permutation circuit organizes the processors automatically as a 1×16 grid. Thus, 15 processors of the 16 can send data *in parallel* to another physical processor that is within the same chip. Let us denote t_I as the time per 32-bit word it takes to move data between processors internally within a chip.

Finally, on every chip one processor must send data to another processor which is on a different chip. Let us denote t_E as the time per 32-bit word it takes to move data externally between processors which are on different chips.

In order to measure the hardware parameters t_P, t_I, and t_E using the ring test we must first measure all the possible overheads. They include the overhead caused by the measurement loop, the CSHIFT call, and the code inside the CSHIFT intrinsic before any data is transferred. Measuring the sum of all of them can be done easily by timing first a CSHIFT call and using a shift distance of 0. Running next the actual test with VP ratios 1, 2, 4, and 8 and using shift distance 1 we get after subtracting the overheads:

$$\begin{cases} t_I + t_E &= 56.3\ \mu s & \text{(VPR = 1)} \\ t_P + t_I + t_E &= 78.1\ \mu s & \text{(VPR = 2)} \\ 3t_P + t_I + t_E &= 120.7\ \mu s & \text{(VPR = 4)} \\ 7t_P + t_I + t_E &= 208.4\ \mu s & \text{(VPR = 8)} \end{cases}$$

Subtracting the equations pairwise and averaging the results we get $t_P = 21.6\ \mu s$ and $t_I + t_E = 56.5\ \mu s$. In order to find out t_I and t_E we have to change our test arrangement slightly. We can use the same ring test but instead of always using shift distance 1 we use distance 16 with the VP ratio 1, distance 32 with the VP ratio 2, distance 64 with the VP ratio 4, etc. This forces all the processors within a chip to communicate externally to the processors within the next chip in the grid, that is, $t_I = 0$ and $t_P = 0$. Because for the external communication between the chips only one wire exists the external communication must be done *serially* [9]. For example, using shift distance 16 and VP ratio 1 the time to send the data of the 16 processors within a chip to the next chip takes $16t_E$. Executing now the test with a few different VP ratios and averaging the results we get $t_E = 26.3\ \mu s$ and thus $t_I = 30.2\ \mu s$. Eq. (2) summarizes our results:

$$\begin{cases} t_P &= 21.6\ \mu s \\ t_I &= 30.2\ \mu s \\ t_E &= 26.3\ \mu s \end{cases} \tag{2}$$

Earlier Levit developed a grid communication model for the CM-2 [9]. However, he considered that the grid communication time is only a function of t_E and t_I. He also measured $t_E = 28.0\ \mu s$ and $t_I = 31.0\ \mu s$ on a CM-2 running at 6.7 MHz. However, based on our measurements (on a CM-2 running at 7.0 MHz) it is obvious that a third component, t_P, must exist. Therefore, we have modified the grid communication model developed by him. We present next only the summary and the interested readers are referred to the paper presented by Levit [9].

If the 16-processor chips are configured with n-dimensional grid geometry $C = \{c_0, ..., c_{n-1}\}$ ($\prod_j c_j = 16$) and each physical processor with a VP geometry $V = \{v_0, ..., v_{n-1}\}$ ($\prod_j v_j$ is the VP ratio) the total time for grid communication of distance $d = 2^p$ (nearest neighbors) along axis i is given by:

$$t_{grid} = \begin{cases} t_E d \prod_{j\neq i} v_j c_j + t_I d \prod_{j\neq i} v_j & \\ \quad + t_P(\prod_j v_j - d\prod_{j\neq i} v_j) & \text{if } d < v_i \\ t_E d \prod_{j\neq i} v_j c_j + t_I \prod_j v_j & \text{if } v_i \leq d < c_i v_i \\ 16 t_E \prod_j v_j & \text{if } d = c_i v_i \\ 32 t_E \prod_j v_j & \text{if } d > c_i v_i \end{cases} \tag{3}$$

The difference to the model developed by Levit is the second formula in Eq. (3) which he considered to be valid always when $d < c_i v_i$. Instead we have added a formula which should be used when $d < v_i$ (communication within the same physical processor exists) and the second formula only when $v_i \leq d < c_i v_i$.

As pointed out by Levit, the communication time is always constant when $d > c_i v_i$. The reason for this is that the NEWS communication takes advantage of the Gray-code ordering of addresses. Using Gray-coding the maximum distance between addresses having a difference of 2^p is two [9, 10].

In Fig. 6 we show the CM-2 communication bandwidth as the function of shift distance. The compiler generates NEWS instructions for the CSHIFT and EOSHIFT functions if the shift distance can be broken into a few power-of-2 distances. We can see from Fig. 6 that NEWS instructions are generated for all the shift distances 1–64 except 63. At shift

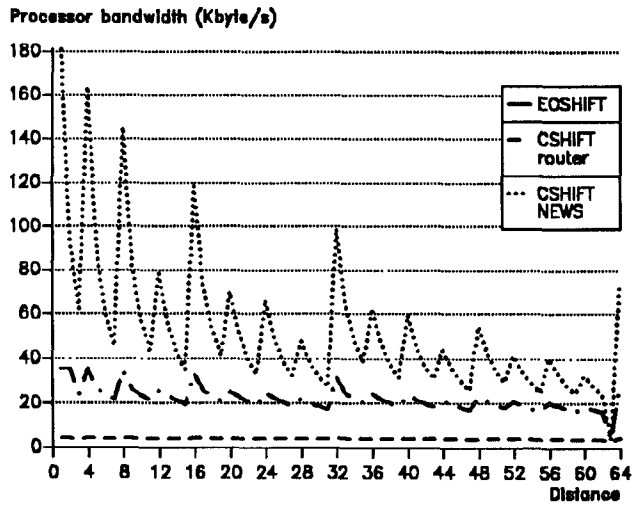


Figure 6: Virtual processor communication bandwidth of the CM-2 as a function of shift distance in a ring of processors ($VPR = 128$).

distance 63 the router network is used. The peak values in the CSHIFT curve represent nearest neighbor communication and they fulfill Eq. (3) quite accurately.

5 Computation with Communication

How well the computation and communication performances of a distributed memory computer system are in balance can be studied by the *balance factor* [3, 11]

$$b = \frac{t_{comm}}{t_{calc}} \tag{4}$$

in which t_{comm} and t_{calc} are defined as the typical time to communicate a *word* between two processing elements and as the typical time to do a generic calculation, such as $a = b + c$ or $a = bc$. A distributed memory system can be defined to be well-balanced if $b < 1$.

Let us study the balance factors for the grid communication, i.e. $t_{comm} = t_{grid}$. If we divide Eq. (3) by the time it takes to do the add operation (from Fig. 3: FPU-32, real*4) we can easily get b as the function of shift distance d, shift dimension or direction i, VP geometry $\prod_j v_j$, and chip geometry $\prod_j c_j$. In Table 3 we have chosen some values for these parameters and calculated the balance factors.

In Table 3 we have got $0.65 < b < 17$. We can see that the balance factor is very sensitive to all the four parameters. Unfortunately, on CM Fortran level the user can not directly modify the VP or chip geometries; this is done by the compiler. On the Paris assembly language level the user has a direct access to configure the geometries, too. We can also notice that when the VP ratio $\prod_j v_j$ increases the

Chip geom. $\prod_j c_j$	*VP geom.* $\prod_j v_j$	d	i	t_{comm}	t_{calc}	b
16×1×1×1	1×1×1×1	1	1	56	49	1.1
16×1×1×1	1×1×1×1	4	1	135	49	2.7
16×1×1×1	1×1×1×1	32	1	841	49	17
16×1×1×1	1×1×1×1	64	1	841	49	17
16×1×1×1	1×1×1×1	128	1	841	49	17
16×1×1×1	4×1×1×1	1	1	121	152	0.80
16×1×1×1	8×1×1×1	1	1	207	290	0.72
16×1×1×1	16×1×1×1	1	1	380	564	0.67
16×1×1×1	32×1×1×1	1	1	726	1113	0.65
4×4×1×1	1×1×1×1	1	1	135	49	2.7
4×4×1×1	1×1×1×1	8	1	841	49	17
4×4×1×1	1×1×1×1	32	1	841	49	17
4×4×1×1	1×1×1×1	128	1	841	49	17
4×4×1×1	2×2×1×1	1	1	313	152	2.1
4×4×1×1	4×4×1×1	1	1	800	564	1.4
8×2×1×1	8×2×1×1	1	1	467	564	0.83
8×2×1×1	16×1×1×1	1	1	406	564	0.72
8×2×1×1	4×4×1×1	1	1	590	564	1.1
4×2×2×1	2×2×2×1	4	2	3366	290	12
2×2×2×2	1×1×1×1	8	3	841	49	17
2×2×2×2	2×2×2×4	4	3	13465	1113	12

Table 3: Balance factors for the grid communication using some shift distances and dimensions and chip and VP geometries.

balance factor b decreases and at some VP ratio the operation becomes balanced. The reason to this is that on higher VP ratios the relative amount of communication compared to computation decreases. For example, using a VP ratio of 16 the sum of all the 16 array or matrix elements within a physical processor can be calculated before any outside communication.

Earlier we have computed balance factors on some distributed memory MIMD computers [3]. Generally speaking, although the CM-2 is unbalanced at lower VP ratios it seems to be much more balanced than e.g. Intel iPSC/860 on which we have measured $17 < b < 430$.

The concept of the balance factor can be extended to operations involving more computation and communication than defined in Eq. (4). A more general way to compare how well the computation and communication performances are in balance is to time the whole operation itself and then the computation and communication separately. Next we studied how well balanced are two such more complex operations, namely scanning and solving Laplace's equation. Scanning is an operation on NEWS grids that combines communication and computation. It can be used to compute e.g. the partial sums or products along a specified dimension in a grid. Special cases of scanning include e.g. finding the sum, product, largest value, etc. over all elements of a row of a matrix. We tested the efficiency of scanning by summing the columns of a $64VPR \times 128$ matrix on an 8K CM-2 first by scanning and then by shifting and adding the elements separately. Since there are 128 columns ${}^2log128 = 8$ steps are needed in the latter method. The results are described in Fig. 7.

If scanning is used b decreases from 11 to 1.4 at VP ratios from 1 to 16. On VP ratios higher than 16 the balance factor is below one and the operation can be considered balanced. If the columns are shifted and summed separately (upper curve in Fig. 7) the relative amount of communication compared to computation does not decrease as in the

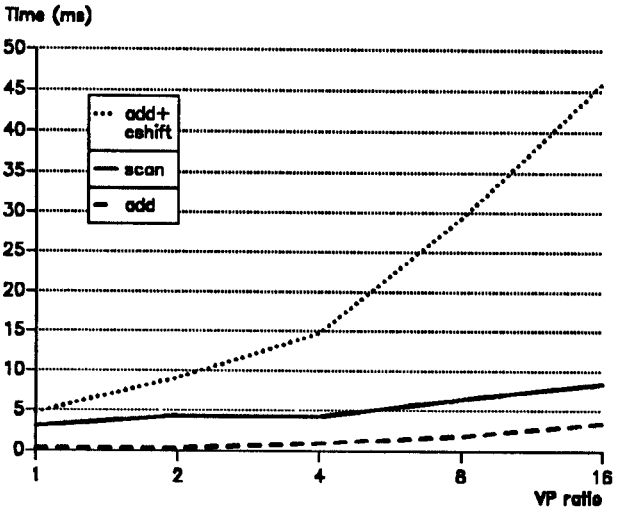
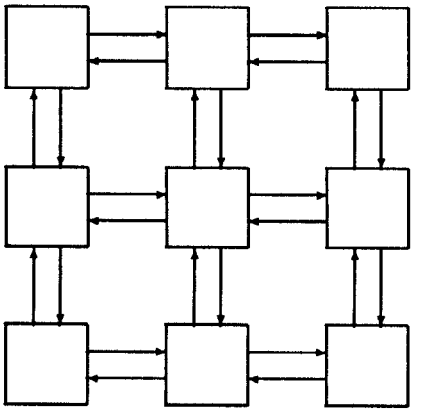


Figure 7: Elapsed time in summing the columns of a $64VPR \times 128$ matrix on an 8K CM-2 (32-bit FPU, Paris execution model).

case of scanning but remains about constant and therefore $13 < b < 26$.

From Fig. 7 we can also calculate that the performance of the scanning kernel is about 27 MFLOPS. Linearly extrapolating this would give about 216 MFLOPS on a full 64K machine. However, possibly this will not be the actual performance because, as we have shown, when the operation includes NEWS communication the elapsed time is a function of VP and chip geometries. Because the user can not set them directly on Fortran level they can be different when the operation is run on a larger machine.

Solving a 2D Laplace's equation using CM Fortran is described in Fig. 8 (the handling of the boundaries is not considered):



```
x = (cshift(x,1,-1) + cshift(x,1,1) +
     cshift(x,2,1) + cshift(x,2,-1))/4.0
```

Figure 8: Solving a 2D Laplace's equation.

The results can be seen in Fig. 9: b decreases from 2.4 to 0.48 at VP ratios from 1 to 128. At VP ratios larger than eight the operation becomes balanced because of the reasons explained earlier in this section. We can also calculate that the performance of the kernel is about 193 MFLOPS. Linearly extrapolating this would give about 1.5 GFLOPS on a full 64K machine.

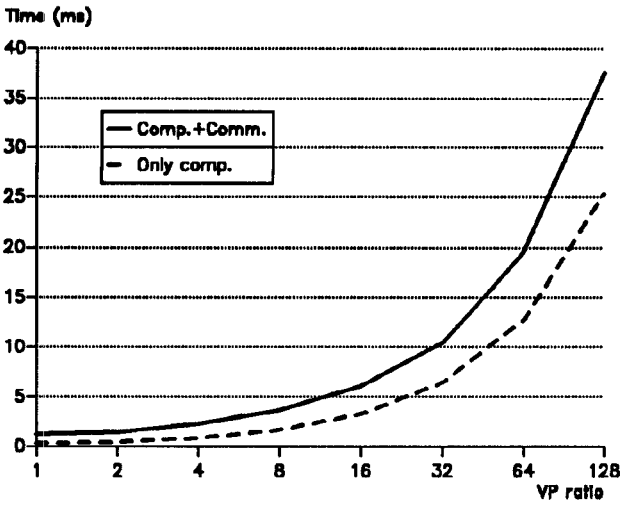


Figure 9: Elapsed time in solving the 2D Laplace's equation using a $64VPR \times 128$ grid on an 8K CM-2 (32-bit FPU, Paris execution model).

6 Conclusions

In this paper we have explained how to test and analyze the basic performance of a massively parallel SIMD computer. The basic computation performance can be studied executing kernels which do not include any communication between processors. The communication bandwidth can be measured configuring the processors using a ring topology and performing a circular shift between all the processors. The communication hardware parameters can be estimated and the grid communication model can be developed. The performance in computation with communication can be analyzed by measuring the balance factors in different kernels.

About the CM-2 used in the study several conclusions can be drawn. First, we found the host systems to be too powerless for the performance of the CM-2. As we have shown its peak performance on Fortran level can be up to 4.4 GFLOPS on a full 64K machine but if the performance of the host is 1–5 MFLOPS the scalar and parallel performances are badly in unbalance. The problem becomes even worse on the CM-200 which, assuming its 43% faster clock frequenzy, will have a peak performance of 6.2 GFLOPS. Also the CM-2 computation vs. router communication performance is very unbalanced and we suggest not to use the router network if it can be avoided. The total grid communication bandwidth of a full 64K CM-2 was evaluated to be 11.8 Gbyte/s. The computation vs. grid communication performance was found

to be unbalanced at lower VP ratios but at higher VP ratios it became balanced. Finally, we found the CM-2 easy to use and potential in highly parallelizable applications.

Acknowledgements

I would like to thank Fredrik Hedman from KTH Stockholm for letting me to use their 8K CM-2a (BELLMAN). Hans Hermann from HLRZ in Jülich helped me to get an userid for their 16K CM-2 (CM-AT-GMD). I am also thankful to Esko Järvinen from CSC Helsinki about his help in making the tests possible on the 32K CM-2 at TMC Boston (CM2-NEARNET). Wolfgang Nagel from Research Center Jülich GmbH (KFA) assisted me to have the results from the Cray Y-MP/832.

The benchmark programs described in this paper can be get freely by anonymous FTP from Tampere University of Technology from the ip-address titan.cc.tut.fi (130.230.23.9). They can be directly applied to test e.g. the CM-200 and with minor modifications to any massively parallel SIMD computer. Hopefully these programs are useful to persons involved in testing and analyzing the performance of future computer systems.

References

[1] *Annual Hardware Buyers' Guide.* Supercomputing Review, Jan. 1992.

[2] J. Helin, K. Kaski: *The Performance Analysis of High-Speed Computers in Scientific/Engineering Environments.* Proc. 1990 Hawaii International Conference on System Sciences, Vol I. IEEE Computer Society Press, Kailua-Kona, HA, Jan. 1990, pp. 268–277.

[3] J. Helin, R. Berrendorf: *Analyzing the Performance of Message Passing MIMD Hypercubes: A Study with the Intel iPSC/860.* Proc. 1991 International Conference on Supercomputing. ACM Press, Cologne, Germany, June 1991, pp. 376–385.

[4] *Connection Machine Technical Summary, Version 6.0.* Thinking Machines Corp., Nov. 1990.

[5] *Connection Machine CM-200 Series Technical Summary.* Thinking Machines Corp., June 1991.

[6] R.W. Hockney, C.R. Jesshope: *Parallel Processors 2.* Adam Hilger Ltd, 1988.

[7] F. McMahon: *The Livermore Fortran Kernels: a Computer Test of the Numerical Performance Range.* Lawrence Livermore National Laboratory Technical Report ULRL-53745, Oct. 1986.

[8] M. Weiss: *Strip Mining on SIMD Architectures.* Proc. 1991 International Conference on Supercomputing. ACM Press, Cologne, Germany, June 1991, pp. 234–243.

[9] C. Levit: *Grid Communication on the Connection Machine: Analysis, Performance, and Improvements.* International Journal of High Speed Computing, 1, 2 (June 1989), pp. 367–381.

[10] S.L. Johnsson: *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures.* Journal of Parallel and Distributed Computing, 4 (1987), pp. 133–172.

[11] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker: *Solving Problems on Concurrent Processors, Vol I.* Prentice Hall, Englewood Cliffs, NJ, 1989.