A MULTI-MICROCOMPUTER SYSTEM FOR MONTE CARLO CALCULATIONS

B. Berg and H. Krasemann
CERN, Geneva, Switzerland

L.O. Hertzberger
NIKHEF, Amsterdam, Holland

ABSTRACT

We propose a microcomputer system which allows parallel processing for Monte Carlo calculations in lattice gauge theories, simulations of high energy physics experiments and presumably many other fields of current interest. The master - n - slave multiprocessor system is based on the Motorola MC 68000 microprocessor. One attraction of this processor is that it allows up to 16 M Byte random access memory.

1. INTRODUCTION

In recent years there has been an increasing interest in Monte Carlo (MC) calculations -- sometimes called "computer experiments" -- for lattice gauge (LG) theories[1]. Typical calculations of this type involve running relatively simple programmes for an enormous amount of time. Consequently there has been a rapidity increasing demand for CPU time in the theoretical physics community. For instance, the CPU time spent by the CERN theory group on the CDC 7600 makes a significant jump by a factor larger than three from 1979 to 1980.

In this article we propose an inexpensive microcomputer (μC) set-up, which is specialized in MC calculations for an interesting class of lattice theories. The system speed is only proportional to the number of variables, but independent of other details of the group interaction on the lattice. In effect it would be comparable to that of an IBM 168 CPU for suitable tasks. Since it operates in the single user mode it will be tremendously faster in real time.

The class of lattice gauge (or spin) theories for which our μC is useful has to match two conditions:

a) the interaction has to be local;

b) the interaction has to be "sufficiently complicated".

The locality condition enables parallel processing , i.e., the upgrading of spatially far enough separated spins can be done independently and simultaneously in time. "Sufficiently complicated" means that the upgrading of a single link takes much more time than an address calculation and data transfer. This condition is met for SU(2) and SU(3) LG theories, but not for discrete SU(2) subgroups or models with even less sophisticated interactions, for instance, the Ising model. In that case the use of a single microprocessor (μP) could, however, still be cost-effective.

Our proposal is based on the MC 68000 for two reasons: address space and the existence of a flexible, sufficiently fast and already working master - n - slave system, the fast Amsterdam multi-processor (FAMP) [2,3]. A different approach is the use of an array processor

like, for instance, the one recently described by Wilson[4]. We will occasionally comment on this approach in appropriate places*). As we ourselves have no experience with an array processor, our comments will be preliminary and we do not present a thorough analysis.

Our paper is organized as follows. In Section 2 we make some more remarks about our approach to parallel processing. Section 3 will introduce the micromachine MC 68000. Section 4 will discuss the details of our parallel system. In Section 5 we will try to estimate the system speed and compare it to that of IBM 168. Section 6 then estimates hardware prices and compares these to the IBM equivalent of 1 year = 200 days running of our system. This comparison is done with a prototype of SU(3) calculations.

A physically very interesting application of our system may be the study of a gauge system with fermions. Furthermore we believe that the same type of parallel processing would be useful for a number of problems in such different fields of science as classical statistical mechanics, plasma physics, evolution of biological populations and atmospheric science, for instance. We would be very interested to get a response from experts in these or others fields, who think that the μC set-up proposed here might be useful for their type of problems. Some final conclusions are drawn in Section 7.

2. SOME REMARKS ABOUT PARALLEL PROCESSING

An elaborate MC calculation on a large 20^4 lattice with the SU(3) interaction which, to obtain some precision, performs 100 upgrading sweeps over the whole lattice at each of 10 points of β (but does not yet contain any measurements), would require $\approx$ 1000 hours of CPU time on an IBM 168. This is outrageous and one has to look for other means. One way out is the use of a very fast processor. This has been proposed and done by Wilson[4] with an array processor which could perform the above-defined task in less than a month. Our idea is similar and different. Similar because we too want to have a computer which, for certain specified tasks like MC calculations, is cheaper than the corresponding time on a big computer. Different because instead of using a fast processor, we want to split the task into many equivalent (but still rather complicated) pieces which can be done in parallel by very cheap (and slower) microprocessors.

For a lattice gauge calculation this splitting is in principle trivial: any upgrading of a single link is sufficiently complicated, i.e., it takes a sufficiently long time that it can be done in parallel. Our system will have a number n of relatively cheap "slave" microcomputers which contain the upgrading routine. A "master" computer will distribute the single tasks, i.e., it will calculate the addresses of all memory locations needed to upgrade one link. It will then distribute these data to one of the slaves and start the slave's programme. Then the master will serve the next slave until after a complete circle it can collect the first slave's answer (the first slave can be any other, which finished its task first) and start a new task.

One obvious condition for this to work is that the time t_1, needed to compute the addresses and to transfer the data, is short against the slaves computing time t_2. The system may contain as many slaves as given by the time quotient,

$$n \leq t_2/t_1 \,. \qquad (1)$$

---

*) Recently we have also learned about the array processor of Ref. 5. We would like to thank K. Bowler for correspondence on this point.

In this system the over-all computing time is essentially determined by the time needed to calculate the necessary addresses and to transfer the data. If the number of slaves does not saturate Eq. (1), the master will partly lie idle. Then the system speed can be increased by adding slaves. For a lattice MC calculation this means that the system speed no longer depends on the details of the group interaction but only on the number of variables and the lattice size.

An SU(2) MC calculation on a lattice spends only $\frac{1}{4}$-$\frac{1}{5}$ of its time in address calculations and data transfers. For SU(3) we estimate this to be down by a further $\frac{1}{2}$-$\frac{1}{3}$. Thus the system speed will be faster than the master speed by ≈ 4 [SU(2)] to ≈ 10 [SU(3)]. On lattices of length L^n (n = 2, 3, 4, ...) further improvements of these factors are possible because all addresses can then be calculated by shifts. In our proposed set-up this factor t_2/t_1 compensates for the lower speed of the single (master or slave) processor. It will at present therefore not be faster than an IBM 168, but, in a small configuration, be even slower in processing speed. In contrast, the array processor used by Wilson[4] is faster than an IBM 168. This does not necessarily mean that the array processor is more cost-effective. An advantage of our system is that one can start to gain experience with very cheap configurations of only a few μP's. The price of an array processor is roughly a factor 20 [30000 $ according to Ref. 4)] higher than the price of a single μP.

Moreover, our proposal offers the opportunity to gain experience in parallel processing. We could imagine that this experience, at a later stage, may become useful in building similar set-ups with faster processors and would thus enable computer experiments on a higher scale of magnitude. Because of the fast progress in microprocessor technology, a considerable improvement in the speed of future generations is possible. The array processor used by Wilson does not really allow parallel processing, but is essentially based on pipelining. Other array processors [e.g. Ref. 5)] allow real parallel processing, but only for very simple steps (e.g. vector operations). This is essentially different from the type of parallel processing we have in mind, which allows independent processing of comparatively very complicated programme sections.

3. THE PROCESSOR

In order to keep the system speed comparable to the IBM 168, the master processor should not be slower than 1/10 of the IBM 168. The necessity of large lattice sizes requires a large address space. An elaborated instruction set is preferable with respect to the future use of high level languages. All these conditions are met with the Motorola MC 68000 [6]. This machine offers:

- high speed [0.5 μs minimum instruction execution time (MOVE)];
- large address space [16 M Byte may contain a 21^4 SU(3) lattice in 16 bit words];
- a very powerful and flexible instruction set which contains integer arithmetic and may be enlarged by the user;
- instructions for multi-processing;
- cross assemblers exist[7];
- preliminary cross compilers exist for Pascal and will come for Fortran[8].

The only seemingly disadvantage of the MC 68000 is the lack of floating point arithmetic. However:

i) the address calculations in the master which determine the system speed are only integer operations (or shifts, etc);

ii) if floating point operations are required, they can be added by special hardware. Later versions with floating point instructions are foreseen. Different floating point ROMs are existing or announced[8]. Similarly, special routines like matrix operations or random number generators should be written and stored in EPROMs. Note that any slowing down in speed caused by (slow) floating point routines can be compensated for by adding more slaves. A later Motorola processor, MC 68020, will be accompanied by a fast arithmetic co-processor for floating point operations[8].

4. THE PARALLEL SYSTEM

In Section 2 we have discussed how parallel processing can be done in a LG MC calculation. Here we will describe the architecture of the system as a variation of the FAMP system. Since the FAMP system is described in detail elsewhere[2,3], we can be rather brief. Our version of the system is shown in the Figure. Its advantages were mentioned before: high modularity, flexibility and simple interfacing to a host computer. The system can be plugged into an RS 232 line between terminal and host computer. A download facility makes use of the host software (cross assembler, cross compiler, etc.) easy. Any task runs in the master, who distributes the subtasks via the dual port memories (DPM) to the slaves. The DPMs appear to the master like any other random access memory. Internal arbitration logic handles simultaneous access demands from the master and slave processor (read or write). At present FAMP system units are being produced at NIKHEF. They will be used in experiment NA 11 at the SPS and experiment UA 1 at the $p\bar{p}$ collider.

For some of the applications we are proposing here, a large amount of memory is required. The best solution is to assemble this memory from commercially available 128, 256 or 512 K Byte modules. As memories with a fast access time (150 ns) are slowly emerging on the market, we propose that our first system contains the 128 K Byte module designed by NIKHEF. Later on we could, for instance, use the announced Motorola units which imply that an interface between the FAMP bus and the Motorola VERSAbus has to be developed.

The CPU modules and the DPMs have to be produced in close collaboration with NIKHEF. Since a system as the one described here demands maintenance, we propose that the final system be built by an outside firm willing to take over the maintenance. One firm has already shown interest.

The FAMP system has an elaborate operating system able to handle the multiprocessor environment[3]. This operating system is written in assembly language and distinguishes a number of layers (kernel, minimum and extended). It is foreseen that the extended operating system will later provide mass-storage handling. The possibility of also installing existing UNIX*) like systems is under investigation.

---

*) UNIX is a trade mark of Bell Companies.

Finally we remark that the FAMP system offers far more flexibility in multi-processing than, for example, the commercially available Motorola VERSAbus system which also allows a multi-processor environment. However, on the VERSAbus all processors use the same bus, which will cause a bottleneck if one wants to increase the number of processors. The FAMP system avoids this bottleneck because any extension is done by separate decoupled bus systems making use of the DPMs.

5. SPEED ESTIMATES

The integer arithmetic of the MC 68000 is roughly ten times slower than that of an IBM 168. For our example of an SU(3) LG MC calculation this makes the system speed comparable to the IBM.

The speed of the slaves is less important. We estimate than ten slaves could saturate the system in an SU(3) calculation when this is done in integer arithmetic. If one prefers real arithmetic it could be implemented as described in Section 3. Here the loss in speed can be compensated for by more slaves which is simple because of the high modularity in the system.

In the configuration we are proposing, the overhead of the operating system is low. This is important because the main load is taken by the master processor.

Finally we remark that for not extremely time-consuming problems even a single μP + memory (without any parallel processing facilities) could be helpful: 24 hours of MC 68000 running time would correspond to two hours of IBM 168 time day for day ... or 200 days × 2(h/day) × × 500(sfr/h) = 200 (sfr/a).

6. COST ESTIMATES

We study three configurations which can all rely on the software of a host computer (compare the Figure):

i) a minimal experimental set-up,

ii) a small configuration with 0.4 M Byte RAM,

iii) a large configuration with 5 M Byte RAM.

All prices are in K ≈ 1000 sfr. The minimal set-up i) is Motorola's design module with 32 KB RAM (5.5 K). It can be plugged in between a terminal (RS 232) and a host computer and is useful for programme development work.

The small configuration has a FAMP master and two slaves as well as memories and is also connected to a host computer via RS 232 interface. It fits into a modified Camac crate (4.5 K). Five CP modules (two spare, 3.5 K each) and three DPM modules (one spare, 1.2 K each) cost 25 K. 400 KB of RAM will cost ≈ 25 K. This gives a system price of ≈ 50 K including spare parts.

This system could carry out an SU(3) LG MC calculation on an 8^4 lattice with a speed five times slower than IBM 168. Per annum it could perform the equivalent of 1000 CPU hours of the 168 which at CERN, for example, would cost ≈ 500 K.

The large configuration has essentially a larger memory (five MB) and more, of the order of 10, slaves. Including the spare parts it needs five Camac crates, 13 CPU boards and 12 DPM modules. This amounts to 82 K. Today the memory would cost ≈ 200 K. This price will decrease further. At today's prices this system will cost 300 K. It is able to do an SU(3) calculation on a 16^4 lattice with a speed comparable to IBM 168, i.e., 5000 CPU hours per annum which at CERN would cost 2.5 millions Swiss francs. The same configuration with only a 0.4 MB memory would only cost 130 K and could do the same amount of calculations on a smaller 8^4 lattice.

It is obvious that the memory is the most cost-intensive factor. It can be foreseen that commercially available modules will become cheap so that the machine can be equipped with 16 MB, e.g., LG calculations on a 21^4 lattice in SU(3) become feasible.

Any of these systems can be completed to be independent by use of a (micro-) computer which is capable of handling high level languages like PASCAL or FORTRAN. An obviously possible choice is Motorola's EXORmacs development system. Others like UNIX have already been mentioned. Such systems would cost 100 to 200 K including a 70 MB mass storage.

7. FINAL REMARKS

We have discussed a version of the FAMP system to be applied to lattice gauge MC calculations. This kind of parallel processing we have in mind is probably not only useful in lattice gauge theories, but also in other applications, where relatively long programme segments can be processed in parallel and which do not require much input or output. Tasks of this form would not use the intelligent (and expensive) peripherals of a big computer and could therefore run much more cost effective on a set-up like our version of the FAMP. Our set-up provides the possibility of increasing the computing power by simply adding hardware (more slaves). For the special case of lattice gauge MC calculations this means that more complicated problems (fermions?) require not more time but only more hardware, which is cheap.

Out set-up is relatively fast; for an SU(3) lattice (pure) gauge MC calculation it is as fast as a big computer. This means that tasks can be run for which no computer centre would provide the CPU time. It is desirable to start and gain experience with a small and cheap configuration of only a few processor boards. A more elaborate starting configuration would cost < 50 K sfr, and it would allow not only studies of the system, but already physics runs, e.g., precision measurements of SU(3) on an 8^4 lattice.

The architecture of our set-up (of the FAMP) is especially interesting because in the future considerable improvements can be foreseen by the fast progress in microprocessor technology.

ACKNOWLEDGEMENTS

We thank B. Martin, P. Ponting and K. Giboni for useful discussions and help, and C. Brugger from Motorola-Genève for providing us with information.

REFERENCES

1) For MC calculations in SU(2) and SU(3) LG theories, a partial reference list is:

K. Wilson *in* Recent Progress in Gauge theories, Cargèse Lectures 1979, Eds. G. 't Hooft et al., Plenum Press, N.Y. (1980).
M. Creutz, L. Jacobs and C. Rebbi, Phys. Rev. D 20 (1979) 1915.
M. Creutz, Phys. Rev. D 21 (1980) 2308.
M. Creutz, Phys. Rev. Lett. 45 (1980) 313.
E. Pietarinen, Helsinki Preprint HU-TFT-80-49 (1981);
C. Rebbi, Phys. Rev. D 21 (1980) 3350.
D. Petcher and D.H. Weingarten, Phys. Rev. D 22 (1980) 2465.
B. Lautrup and M. Nauenberg, Phys. Rev. Lett. 45 (1980) 1755.
G. Mack and E. Pietarinen, Phys. Lett. 94B (1980) 397.
J. Groeneveld, J. Jurkiewicz and C.P. Korthals Altes, Utrech Peprint (Dec. 1980).
B. Berg, Phys. Lett. 97B (1980) 401.
G. Bhanot and C. Rebbi, Nucl. Phys. B(FS), to be published.
B. Berg and J. Stehr, Z. Physik C, to be published.
J.D. McLerran and B. Svetitsky, Phys. Lett. 98B (1981) 195.
J. Kuti, J. Polonyi and K. Szlachanyi, Phys. Lett. 98B (1981) 199.
R.C. Edgar, L. McGrossen and K.J.M. Moriarty, University College London Preprint (Nov. 1980).
C.B. Lang, C. Rebbi, P. Salomonson and B.S. Skagerstam, CERN Preprint TH 3021 (1981).
J. Engels, F. Karsch, I. Montvay and H. Satz, Bielefeld Preprint, BI-TH 81/05.
A. Billoire, G. Lazarides and Q. Shafi, CERN Preprint TH. 3064 (1981).

The literature is exponentially increasing. Further references can be found in the given papers.

2) L.O. Hertzberger et al., FAMP hardware, these proceedings.

3) D. Gosman et al., FAMP operating system, these proceedings.

4) K.G. Wilson, Experiences with a Floating Point Systems Array Processor *for* the Series in Computational Physics on Parallel Computations. Preprint, Cornell CLNS-81/477 (1981).

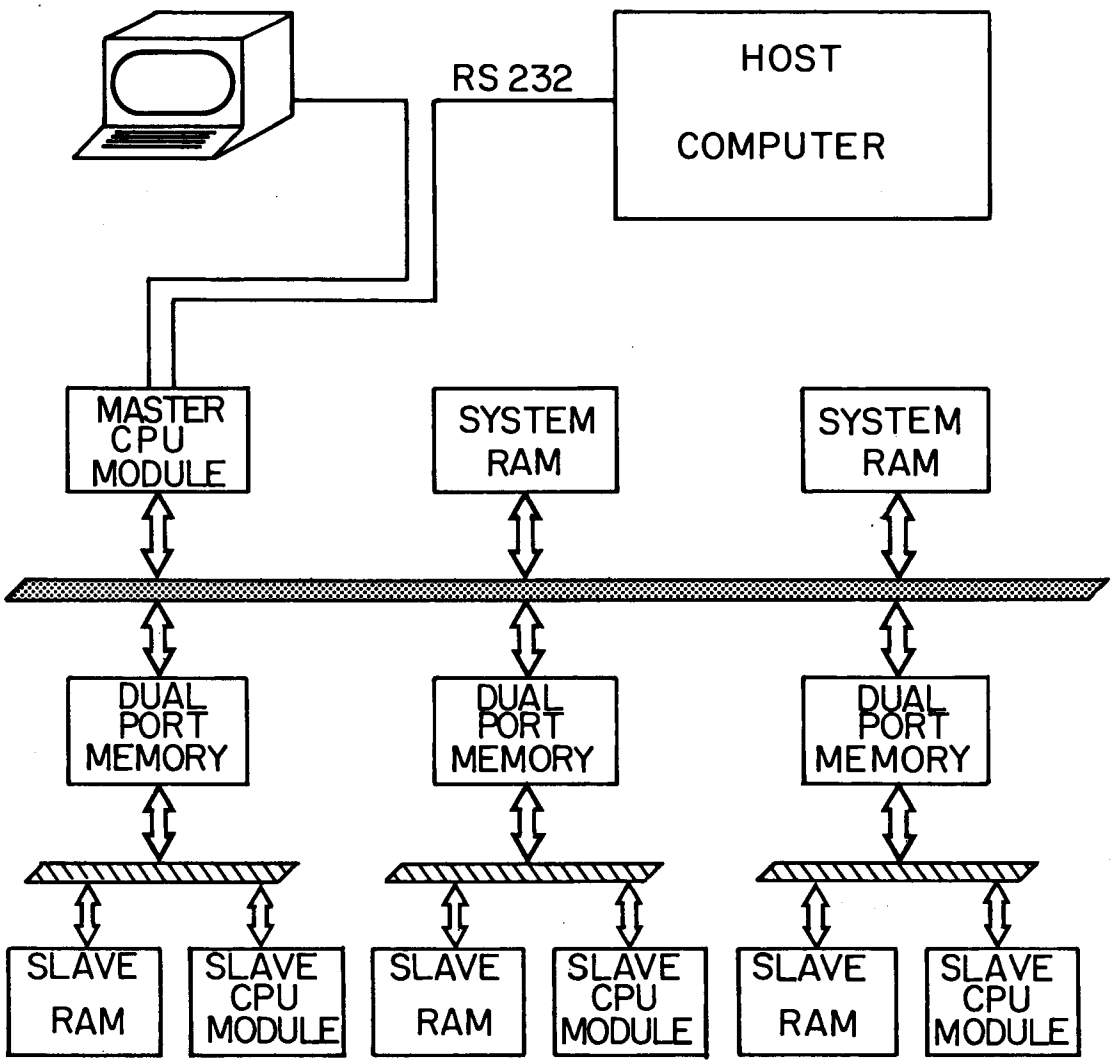
5) G.G. Scarrot, ICL Tech. J. 1 (1978) 35.
R.W. Gostick, ICL Tech. J. (May 1979), p. 116 ff.

This array processor needs an ICL computer.

6) Motorola, MC 68000 16-Bit Microprocessor, User's Manual (1980).

7) Motorola, Cross Macro Assembler, Reference Manual (1979).
Motorola, Resident Structured Macro Assembler, Reference Manual (1980).
V. Eicken, Cross Macro Assembler for the MC 68000, CERN Computer Library.

8) Motorola Seminar "Microcomputer Forum" (1981).



The master-slave system with n = 3. The system communicates via dual port memories (DPM), which lie in the master's address space. In addition the master can address 16 MB RAM on its bus.