A Multiple Microprocessor System for CPU-bound Calculations

L. M. MACKENZIE,* A. M. MACLEOD AND D. J. BERRY
Department of Natural Philosophy, University of Glasgow, Glasgow G12 8QQ

This paper describes a multiple microprocessor system, under development at Glasgow University, for application to calculations arising in the theory of the Nuclear Shell Model. It is the intention of the authors to discuss the architecture rather than the operation of this machine, and to concentrate particularly on design features which will allow future expansion of both capability and applicability within the range of such computations.

Received September 1985

1. INTRODUCTION

In recent years there has been a growing awareness of the potential provided by massively parallel systems to bridge the frustrating gap between computer performance and the computational demands of present-day scientific research. While performance limitations have tended to set fairly tight constraints on the applicability of integrated microprocessing units to highly CPU and memory-intensive concurrent computations,[1] VLSI fabrication techniques have increased the processing power of such devices by up to two orders of magnitude in the last decade. In consequence, many of the microelectronics manufacturers are now acutely aware of the potential of their latest products to influence the designs of high-performance computer architectures, as witnessed, for example, by the marketing of the Inmos IMS T424 'transputer', a 32-bit single-chip microcomputer proclaimed by its designers as an ideal building block for extensive multiprocessor assemblies.[2]

While significant national programs are currently directed at the development of general purpose 'fifth-generation' parallel architectures, the performance of 'state-of-the-art' VLSI technologies can be brought to bear on intractable numerical or logical calculations by means of more specialised, but relatively low-cost, modular multiple microprocessor systems, dedicated to the solution of particular classes of problem. This approach has several important advantages as follows.

(1) The performance attained can be very high, even in terms of absolute comparison with contemporary supercomputers, and is subject to incremental improvement when required.

(2) Since the machine has the character more of a laboratory-based super-calculator than a computer installation, comparisons of absolute performance are in any case grossly pessimistic. The effective processing capacity (power/availability product) at the disposal of a research group can be several orders of magnitude greater than that provided by an annual allocation on a centralised supercomputer installation.

(3) Running and maintenance costs of a well-designed machine are so low that it should be possible to recoup the initial construction outlay rapidly in saved mainframe time. Modularity, in particular, if effectively exploited, facilities rapid repair of hardware failures.

* Now at Department of Computing Science, University of Glasgow.

The difficulties involved in an undertaking of this kind, however, are not negligible. There is a fundamental requirement for a flexible and extensible hardware design, optimally adaptable within often rigid financial and operational constraints. Adaptability is important since future perturbations or extensions to a method cannot always be foreseen. There is, in addition, the potential bonus that an architecture with a sufficient degree of inherent generality might form the basis of similar dedicated systems devoted to other, perhaps quite unrelated, computational problems, thus reducing research and development effort in future undertakings.

The idea of machines dedicated to specific problems or classes of problems is not, of course, new, and has found favour especially in theoretical physics.[3] The authors are interested in the design of systems of this kind and have, in particular, been concerned with calculations of the type arising in the theory of the Nuclear Shell Model. The remainder of this paper will outline a design for a *Nuclear Shell Model Processor* which exemplifies the above approach, and which has, in fact, already been partially implemented in an ongoing development project.

2. THE SHELL MODEL PROCESSOR: WHY A MULTIPLE MICROPROCESSOR SYSTEM?

In quantum mechanics, each observable quantity (position, momentum, energy, etc.) is represented as a linear operator acting on a *configuration space* of state vectors, corresponding to the allowable 'states' of the target system. The Nuclear Shell Model involves a study of such quantum mechanical configuration spaces of very large dimension. State vectors with as many as 10^6 elements, and matrix operators with 10^{12} entries are generated by nuclei of only medium mass number. An ideal Nuclear Shell Model Processor should be capable of performing a range of relevant computations including the determination of the eigenvalues and eigenvectors of quantum operators, density matrix elements of state vectors, expectation values of observables, etc. The calculation of the energy eigenstates (the eigenvectors of the energy operator) of a given nucleus, in particular, is at the same time both crucial to the theoretical development and exceptionally computationally demanding, involving the evaluation and diagonalisation of a symmetric matrix operator, the *Hamiltonian*, acting on the nuclear configuration space. The *Lanczos* algorithm is now accepted as the standard method of

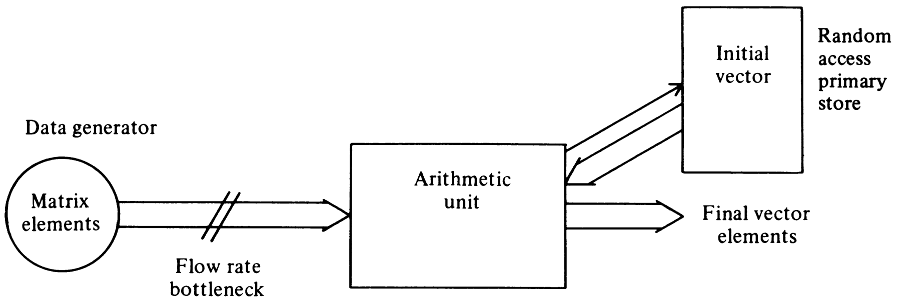

Figure 1. Multiplication of a large dimensional vector by an irregularly sparse matrix

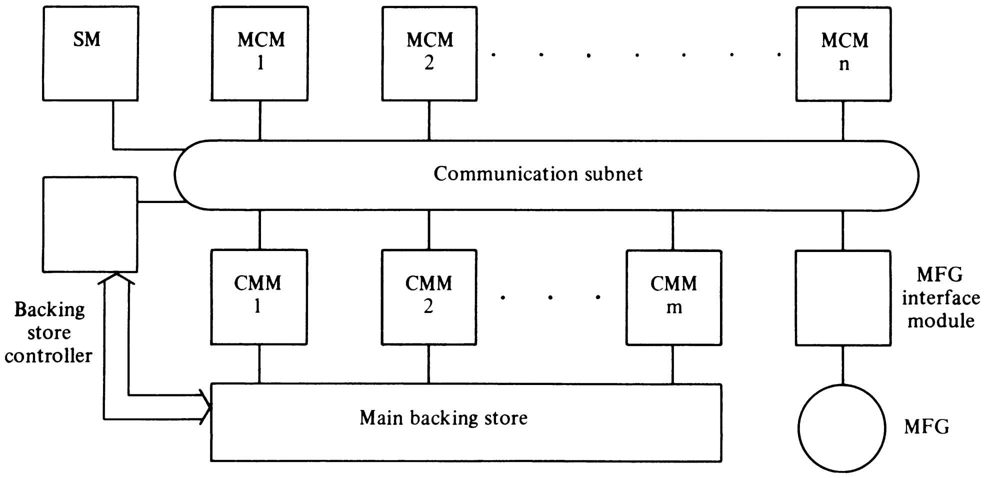

Figure 2. Shell model processor station (MMPU). Note: subnet provides a communication service between any pair of modules interfacing to it.

tri-diagonalising the Hamiltonian matrix in an angular-momentum uncoupled representation, since the approximations to the lower eigenvalues converge after only relatively few (say 100) iterations.[4] The capacity to execute this algorithm is, therefore, a necessary, but by no means a sufficient, condition for a successful Shell Model processor.

A Lanczos iteration involves several matrix/vector operations, of which the most time consuming is the multiplication of a nuclear state vector by the Hamiltonian. While, at least in principle, modern VLSI technology makes the construction of a powerful and dedicated parallel matrix-vector multiplier a fairly straightforward undertaking, the capability of a machine of this kind is severely constrained when the arrays involved are very large and, as in this case, irregularly sparse. The matrix, with perhaps more than a thousand million real entries, cannot be held in primary storage, so that there is a practical limit to the rate at which operands may be fed to an arithmetic processor (see Fig. 1).

There are two alternative approaches to this problem.

(1) *Matrix storage*. The matrix can be computed once and held on disk, being retrieved and fed to the arithmetic unit during each iteration.

(2) *Matrix generation*. The matrix can be generated in real time during each iteration, without ever being actually stored.

Since the number of elements is so large, the former approach would require some tens of gigabytes of on-line, fast secondary storage, and the technique is inevitably extremely expensive; indeed for large calculations it is probably not feasible. Matrix generation, on the other hand, appears to have a greatly superior overall ratio of performance to cost, but requires substantial additional computational power. The authors have developed and tested a prototype generator, the MFG (Matrix Format Generator), which combines a high-performance MC68000 microcomputer and a dedicated ECL hardware accelerator, to produce in real time partial descriptions of the Hamiltonian matrices of sd-shell nuclei (i.e. those with between 9 and 20 protons and between 9 and 20 neutrons) identifying the positions, but not the values of all non-zero elements. The problem of evaluating these elements is highly parallel in nature, but has an asynchronous heterogeneous nature which demands the versatility of a multiple CPU machine rather than, say, an array processor. A multiple microprocessor system of the kind discussed above is an ideal solution in this situation. Although it does not exclude a storage approach for smaller calculations, it can provide the flexibility and performance, at suitably low cost, to run generation algorithms.

The Shell Model Processor project is seen as consisting of two phases. Phase I, now approaching completion, is a practical feasibility study, involving the construction of a 'pilot' multiple processor, driven by the MFG, and capable of handling calculations with up to 32 single-particle nuclear orbitals. Phase II will require the production of a significantly more ambitious machine with up to 4 times this orbital capacity. The Phase II system, as currently envisaged by the authors, is

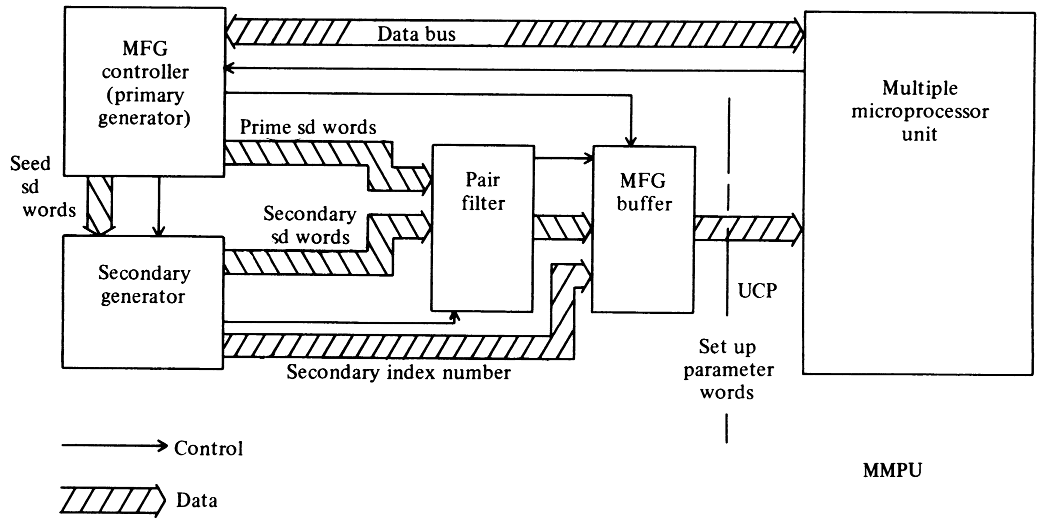

Figure 3. Matrix format generator. Logical block diagram

essentially an extension of the existing Phase I processor; in the following discussion the complete extended architecture will be described, indicating those areas not applicable to the prototype.

3. GENERAL ARCHITECTURE

The multiple microprocessor system developed by the authors for application to Nuclear Shell Model calculations is based on the modular architecture illustrated in Fig. 2. The fundamental building block is a self-contained station called a *Multiple Microprocessor Unit* (MMPU) which has stand-alone capability but can be linked to other stations, providing scope for horizontal expansion should it ever be desired. The present work will be (even in Phase II) restricted to the construction of a single MMPU which should have performance characteristics more than adequate for projected requirements. Within an MMPU, control resides with a single *Supervisor Module* (SM), which coordinates the activities of a number of general-purpose processing elements called *Microcomputer Modules* (MCMs), each an independent computer in its own right. All the MCMs may randomly access the shared *Central Memory Modules* (CMMs), which provide bulk storage for global data such as the vectors in a Lanczos iteration. Additionally they may obtain parameters from an external generator which can act as a data driver for internal MCM processes. This generator, which could be a massive secondary storage facility or a front-end processor, acts as the source of matrix elements during the Lanczos matrix-into-vector step. The present arrangement uses the prototype MFG in this role (Fig. 3), and is expected to continue until the system is required to execute calculations involving nuclides with active pf shells.

The fundamental feature of any multiple processor system is its communications *subnet* to which all its constituent processors (*hosts*) interface. The subnet's properties are defined by the system interconnection topology and, for many applications, determine the absolute limits of performance. Conventional multi-microprocessors fall squarely into Flynn's MIMD category:[5] systems consisting of many processors running what are essentially autonomous but, in general, intercommunicating processes. It is now widely accepted that, for large machines in this class to be successfully implemented, individual processors must be endowed with local resources (especially memory) so that the global subnet is loaded only when necessary. In particular, CPU references to the instruction stream and to local variables can be removed from the subnet altogether, significantly reducing the *utilisation ratios* of individual CPUs (i.e. the ratio of subnet bandwidth required by a processor to total bandwidth required by that processor).

Many structures have been proposed for multiple processors: for example the crossbar switch in Carnegie Mellon's C.mmp;[6] shared memory in UMIST's CYBA-M;[7] a binary 'n'cube' in Caltech's COSMIC CUBE;[8] linked buses in Cm*, etc.[9] The MMPU subnet is based on a simple multiple shared bus. Despite, or perhaps because of, their simplicity, bus-oriented subnets have several significant and desirable natural properties.

(1) The subnet does not require internal 'intelligence'. The routeing, congestion and flow control problems characteristic of, for example, packet-switched networks, are eliminated or, more precisely, are reduced to the level where they can be handled by fast hardware. Bus hardware, in general, is simple, fast, reliable and relatively easy to debug.

(2) The subnet is itself symmetrical in the sense that any node can reach any other directly with no routeing delay. The symmetry makes it particularly easy to interface special devices to the system, such as, for example, shared memory modules or special processors.

(3) The subnet is flexible in that total available bandwidth can be divided in any desired way amongst hosts. Thus a specialised host requiring, say, heavy bursts of traffic (e.g. an array processor) can be allocated as much bandwidth as arbitration protocols permit, up to, of course, the total available limit.

(4) The structure makes not only point-to-point but also broadcast transfers extremely easy to effect. The latter are often very useful where globally significant information has to be transmitted, or where multi-host synchronisation is desired.

Although these advantages are clear, bus structures have tended to be regarded as rather restrictive. The total

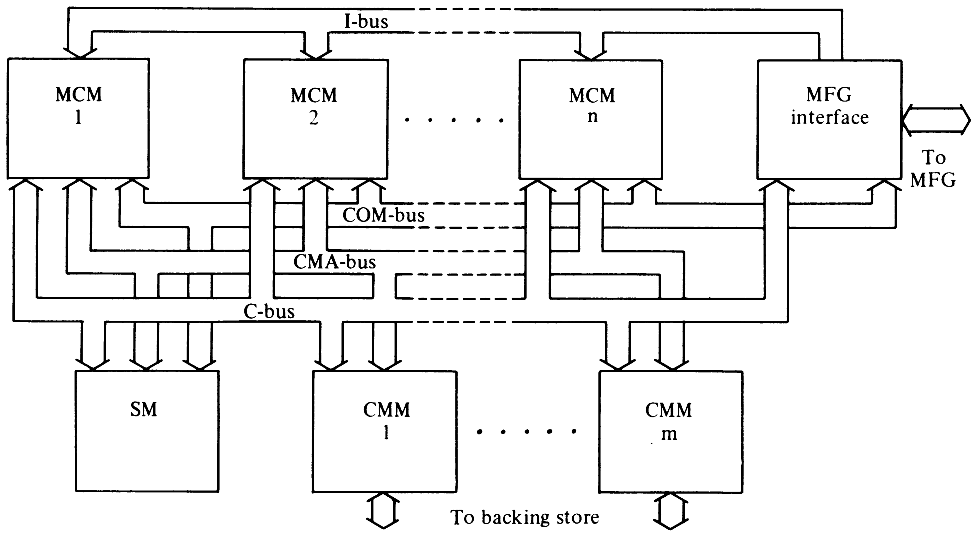

Figure 4. Block diagram of MMPU (Phase II)

available bandwidth, B, on a given bus, is a characteristic upper limit determined by the technology. No matter how large B is, there is a point beyond which the system cannot grow without encountering overloading (below this point the modularity of bus systems is excellent). If B is high enough this objection is more a matter of aesthetics than a serious practical worry (any parallel machine designer would like to believe that his architecture is infinitely extensible), but until recently the problem has been precisely that values of B have been too small.

Modern bus specifications, however, offer bandwidths of between 30 and 40 megacycles/s: notably the IEEE 896 *Futurebus*[10] and the NIM *FASTbus*,[11] now being adopted as IEEE Standard 960-1984. Using advanced ECL drivers it is probably already feasible to achieve a transfer rate of 50 MHz, so that, say, a 32-bit bus with a bandwidth of 150–200 Mbytes/s is not by any means inconceivable. If restricted to essential data transfers (i.e. no code or avoidable data transfers) such a bus can satisfy the peak requirements of at least 50–100 high-performance microprocessors (e.g. NS32032 or MC68020). Hence, although the objection remains valid in the sense that a pure bus-based system is not feasible for massive parallel systems with thousands of processors, an extremely powerful flexibly coupled multiprocessor based on message passing, shared memory or both, can be constructed. Such machines could, of course, form 'supernodes' on a more extensive 'supersubnet'.

The MMPU uses four buses (Fig. 4) to provide the necessary interconnection between its host components. (As yet only two have been implemented in the pilot machine, but a reduced CMA-Bus will be added eventually.) Before discussing these individually, a general comment might be helpful. The Phase II MMPU subnet is intended to provide bandwidth requirements well in excess of those currently projected as necessary for the most powerful Phase II module designs, which might each have, say, 20–30 times the performance of a Motorola MC68000L8. The Shell Model application requires a fairly low subnet utilisation ratio for each processor, so that in fact, in this case, the communications structure described below is powerful enough to support processing technology almost an order of magnitude faster than the best available today. It is therefore feasible that a future (Phase III?) Shell Model Processor could use virtually the same subnet, but support, say, 20 modules, each with sufficient processing power to execute 100 million operations/sec.

(1) C-Bus (Command Bus) is the primary MMPU communication highway, connecting all modules together and intended to carry system-level command and control messages. It is also used in the pilot machine to transmit bulk data and process code, although this function will probably be largely subsumed by COMbus in the Phase II implementation. In order to obtain access to a pool of available off-the-shelf hardware, it was decided to base C-Bus on the now widely accepted Motorola/Mostek/Signetics/Thomson VMEbus.[12] Although the performance of this standard is moderate by comparison with the structures discussed above (< 40 Mbytes/s), it was felt that the C-Bus function could be adequately supported and that compatibility with an industry standard was consequently a more important consideration. VMEbus includes 32-bit data and address buses, four levels of daisy chained arbitration and seven levels of interrupt. Data transfer occurs via a fully interlocked asynchronous handshake.

The authors have augmented the standard VMEbus specification in two ways intended to enhance multiprocessor support.

(*a*) A *Bus Broadcast* facility has been included, enabling a suitably privileged master to write data simultaneously to any subset of MCMs.

(*b*) The lowest bus request/grant priority level BR0* BG0IN*/BG0OUT* uses a decentralised daisy-chain grant protocol which removes the position-dependent prioritisation inherent in the normal VMEbus system (none the less retained for levels BR1* – BR3*). MCMs, which are by definition isomorphic modules, share this line and thus have virtually equal priority on C-Bus.

Despite these changes, upward compatibility with VMEbus is maintained by identifying C-Bus-specific accesses using the VME Address Modifier lines. Thus a standard VME card is entirely compatible with custom-

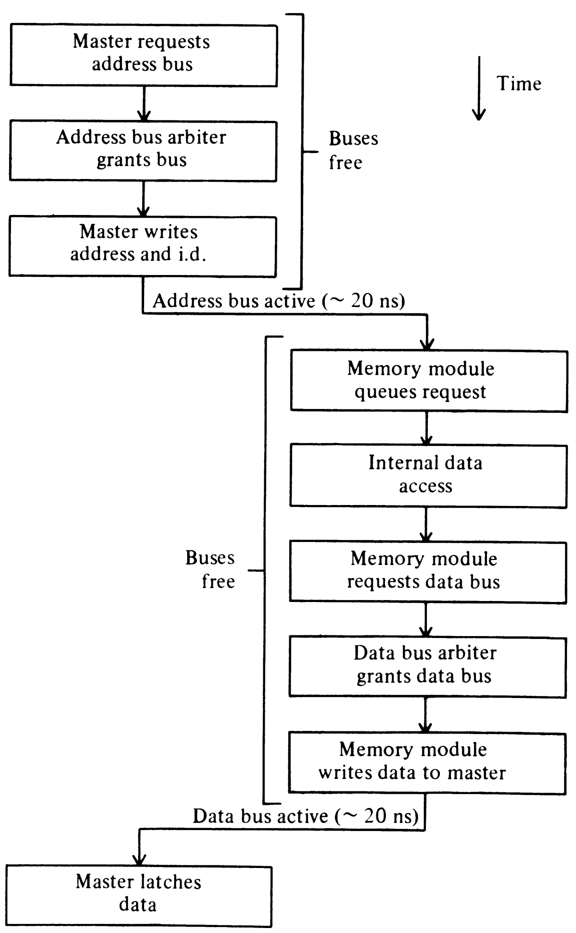
Master requests address bus
Address bus arbiter grants bus
Master writes address and i.d.
Buses free
Time
Address bus active (~ 20 ns)
Memory module queues request
Internal data access
Memory module requests data bus
Data bus arbiter grants data bus
Memory module writes data to master
Buses free
Data bus active (~ 20 ns)
Master latches data

Figure 5. Concurrent address/data transfer on CMA-bus

ised MMPU modules designed to accord with the C-Bus enhancements.

(2) I-Bus is a dedicated data bus which provides MCMs with a high-speed communications route to as many as 32 single-address devices and, in particular, to the Shell Model Processor's MFG front end. Advanced Schottky bus drivers enable transfer rates in excess of 120 Mbytes/s to be attained on a 56-bit data pathway. This is considerably in excess of requirements, in keeping with the philosophy outlined above: in fact a fully populated Phase II machine would require an I-Bus bandwidth of no more than 20 Mbytes/s. Data transfer is again asynchronous, governed by a four-edged handshake; arbitration is single priority and is accomplished by means of the same decentralised daisy-chain protocol employed for C-Bus priority level 0.

(3) CMA-Bus is a bidirectional 64-bit shared data pathway designed to support fast random read/write cycles, over a 2 Gbyte range, for transfer of operands between MCMs and CMMs. The incorporation of a bus devoted to such transactions is necessary to free the system from the constraints of a conventional shared-bus architecture. The specification allows the data and address buses to operate concurrently on independent transfers so that very high performance is attainable (Fig. 5). With ECL interfaces, bus cycle times of less than 30 ns, and fully pipelined arbitration a bandwidth potentially in excess of 300 Mbytes/s would be attainable (Phase II requirements are projected as < 50 Mbytes/s). To support these access rates the CMM address space would need to be interleaved between several (say 16) independently accessible memory banks distributed over a number of CMMs: clearly, an ability to queue incoming read and write requests for later service would be required. Data determinacy during multiple read–modify–write operations on CMA-Bus will be preserved by means of hardware-driven *lockout* flags which will protect each CMM location.

(4) COM-Bus is a high-speed message-passing inter-MCM link proposed for the Phase II machine, with a maximum bandwidth of up to 200 Mbytes/s, shared on a cycle-by-cycle basis, in such a way as to allow many concurrent private conversations (a broadcast facility could easily be accommodated). High bandwidth communication between MCMs is not required during any sd-shell applications, and anticipated needs in the pilot system can easily be satisfied by C-Bus alone or, exceptionally, by means of a Central Memory 'mailbox' system.

The MMPU modules are inevitably subject to design constraints imposed by the requirement that they interface consistently to the protocols of the subnet. In the next section the three major categories of applicable design constraint will be briefly examined: hardware-imposed, C-Bus-imposed and function-imposed. Despite these restrictions there is still almost total freedom in the details of internal module design, maintaining flexibility and facilitating the replacements of older units as technological improvements permit. The following discussion is necessarily, however, incomplete and must draw heavily on experience of the Phase I implementation of the subnet.

4. DESIGN CONSTRAINTS SPECIFIED BY MMPU

4.1 Functional constraints

The functional constraints imposed on a module are dictated by the operational requirements of its role within the system. Although the Shell Model Processor could legitimately be regarded as a 'calculator' it would be misleading to restrict a discussion of its target problem set to the Lanczos iteration for, as indicated earlier, not only is this set currently more extensive (e.g. calculation of density matrix elements, expectation value of quantum observables, etc.), but in addition there is the need, as already emphasised, for inherent functional flexibility. An MCM, for example, must be capable of performing a large and, indeed, still incompletely specified list of widely differing tasks.

For these reasons it is important to construct a system which is configured to run a range of software packages, including future user-generated programs. This kind of freedom is only realistically available if an appropriate independent operating system is installed to provide a user/machine interface. The MMPU is capable of providing hardware support for operating systems ranging from the centralised to the distributed, as desired by the user. For example, the Supervisor Module could be programmed to exercise tight control over all system activities in a strictly hierarchical manner, or to intervene only when asked for assistance by an MCM.

In the case of the Shell Model Processor, since the users are liable to be themselves experienced programmers, and since the range of applications is liable to be relatively restricted, the operating system can be fairly unsophisticated. As envisaged at present (current software packages for the MMPU have only very limited operating system support), it will consist essentially of a supervisory executive running in a multiprogrammed environment on the Supervisor Module, overseeing a series of distributed local *kernels*, each physically resident on one of the slave modules. User processes will be assigned by the executive, arbitrarily or by user specification, to given modules, where they will run under the control of the local kernels. The operating system will handle all interprocess, and hence all interprocessor, communication, task scheduling and resource management.

User programs may be written directly in assembler, or in a high-level language provided with an appropriate library of system call procedures (the authors have done this for Pascal). In either case, operating system functions are ultimately accessed via software-generated exceptions, following a predefined protocol (e.g. in the present rudimentary system, calls are made by means of the 68000 TRAP no. 15 instruction). Many hardware resources, including the subnet and local peripherals (co-processors, I/O lines, etc.) are only available to a processor running in system mode, so that, for example, one user process wishing to pass data to another must trap to the local kernel. System processes can, of course, access the hardware directly. During a Shell Model iteration the Supervisor Module functions mainly as a watchdog, responding to interrupts generated by other modules in need of central services (in normal Shell Model processing such interrupts are typically initiated by error conditions). It is also, of course, responsible for overall coordination of the system as an iteration is scheduled or terminated, and for providing a user interface to the operator.

Functionally, each *active module* (i.e. each module capable of running a user process) will be identified to the operating system as either an MCM (general-purpose) or a special-purpose unit. Unassigned processes will only be run on MCMs, but at initiation time the operator can declare that a newly installed process is to run on a specified module.

Since modules may vary widely in their internal topology, and may indeed support several microprocessors, it will clearly be necessary to define software interfaces governing communication between the local kernel and the external operating system, consisting of other local kernels and the supervisory executive. Once this is done, internal kernel design can be tailored to suit the architectural requirements of any given module.

4.2 C-Bus constraints

The overall processor-memory description of any module must conform to the constraints imposed by the C-Bus addressing structure. Since C-Bus supports a 32-bit address bus, a processor with C-Bus master capability, when in operating system mode, will view the physical system as a 4 Gbyte block, certain regions of which may be restricted from access either by Supervisor-level protection, or by target module memory management. Of this total physical address space, each active module is assigned 128 Mbytes which are internally accessible to on-board processors without the use of C-Bus.

Up to 20 active modules may reside within an MMPU, so that a total of 2.5 Gbytes of the system space are reserved for their use. The remaining 1.5 Gbytes are divided in any appropriate manner between modules such as CMMs or other dedicated units. The 128 Mbyte block of the system address space allocated to an active module, called its *Primary Module Map* (PMM), does not necessarily contain all addressable on-board devices. It is also permissible for processors to use locations which may be switched out of the PMM or, indeed, which are inaccessible to it by direct random-access operations. There is no constraint on the number of processors which may reside within a module. If there are several, they may be organised in any desired manner, for example hierarchically, functionally or with co-equal access to on-board resources (see Section 5).

4.3 Hardware constraints

At the hardware level the only significant constraints are that each module should satisfy the electrical loading and signal protocols specified for each bus interface which it supports. Every module is interfaced to C-Bus but only MCMs and CMMs to CMA-Bus, only MCMs and peripheral interface modules to I-Bus and only active modules to COM-Bus (Fig. 4). Although an MCM must interface to the 4 system buses, only C-Bus can act as an extension of the processor's local bus. The CMA-Bus, I-Bus and COM-Bus interfaces are specially designed *pre-fetch buffers* (*PFBs*) which can conduct memory cycles independently of, and in parallel with, the on-board MPUs.

Also, there is a practical requirement for some degree of low-level software compatibility between modules. This implies a need to link the MMPU architecture to a microprocessor architecture which essentially combines currently available high performance with projected upward-compatible 32-bit machines. The authors have selected Motorola's M68000 family as, in their view, providing the optimal mix of these qualities.[14] The MMPU as presently implemented is configured to support the recently announced MC68020 microprocessor,[13] but the prototype modules which are already installed are based on the proven MC68000 and MC68010 MPUs.

5. MCM DESIGNS

To indicate the practical realisation of the concepts discussed above, it might be helpful to give some indication of the nature of the hardware which has been designed for the Shell Model Processor project. The MCM is not only the major determining factor in fixing the limits of real system performance, but it is a paradigm which can be used as a basis for the design of other active modules, and its internal architecture might be expected to be particularly instructive. A number of MCM designs (Figs 6, 7, 8) have been studied seriously. These are monoboard processing elements of increasing computational power and can perform well over a wide range of applications. However, they are tailored to tackle calculations of the type arising in the theory of the

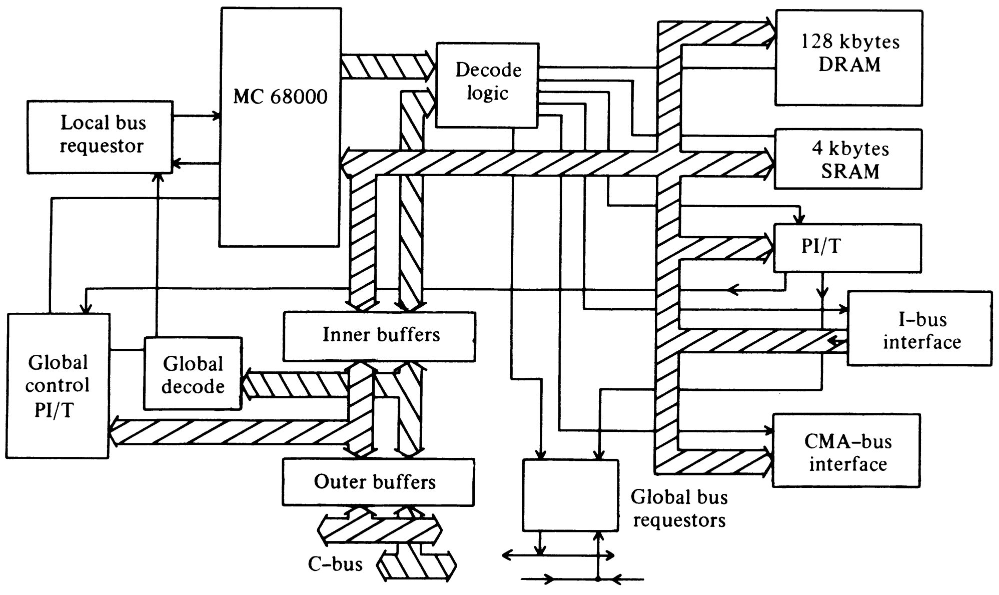

Figure 6. MCMI block diagram

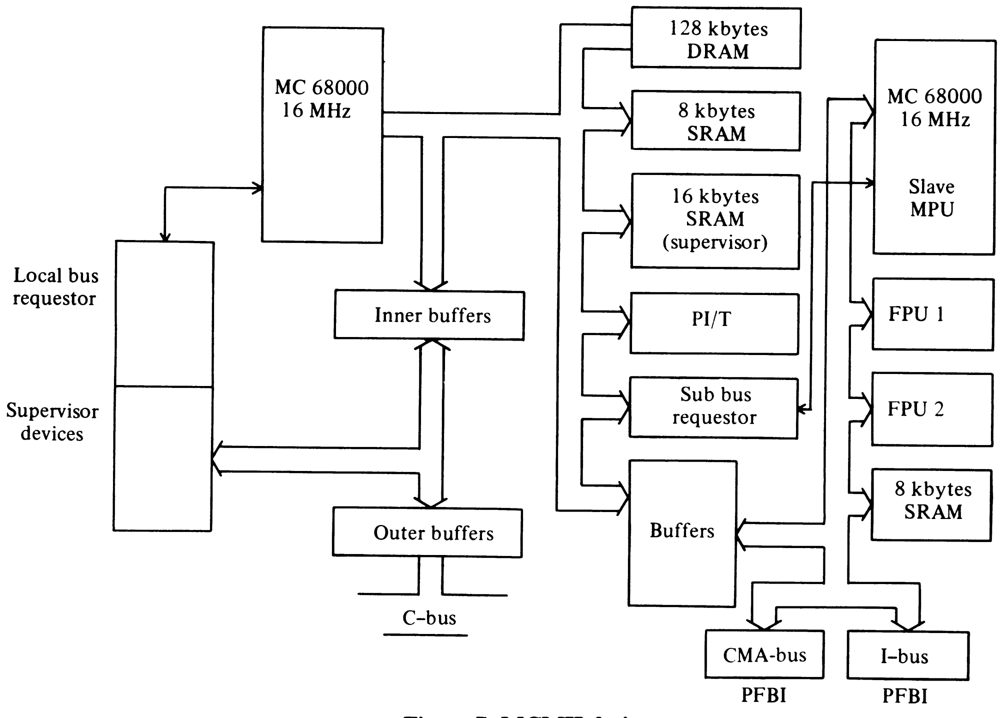

Figure 7. MCMII design

Nuclear Shell Model, and performance figures quoted must be treated accordingly.

MCMI (Fig. 6), built as part of an early feasibility study (1982), was designed rather to test system concepts than for optimal performance. The local bus topology is simple and supports only one processor, an 8 MHz MC68000, but all on-board devices are dual-port with respect to C-Bus. As with all its successors there is no on-board firmware, and all system code is loaded by the SM at initiation time into protected areas of RAM. This gives a tremendous amount of inherent flexibility, allowing dynamic tailoring of a module kernel and assisting enormously in its development and testing.

The MCMII design (Fig. 7), now operationally tested, is intended to act as an advanced prototype capable of providing processing power adequate for extensions of the calculations to higher nuclear shells. The module is hierarchically organised around a single *master* processor, an enhanced-performance MC68000 running at 16 MHz (a steady 1–2 MIPs capability). An 8 Kbyte block of very fast static RAM allows the 16 MHz processor to execute a memory access (read or write) in 250 ns (no wait states) and is intended to hold time-critical program sections and frequently accessed variables. The master MPU is also provided with 128 Kbytes or 512 Kbytes of local bulk memory which runs with 4 wait states (375 ns cycle time). A second 16 MHz 68000 acts as a slave on a local sub-bus to which are directly interfaced the I-Bus and CMA-Bus

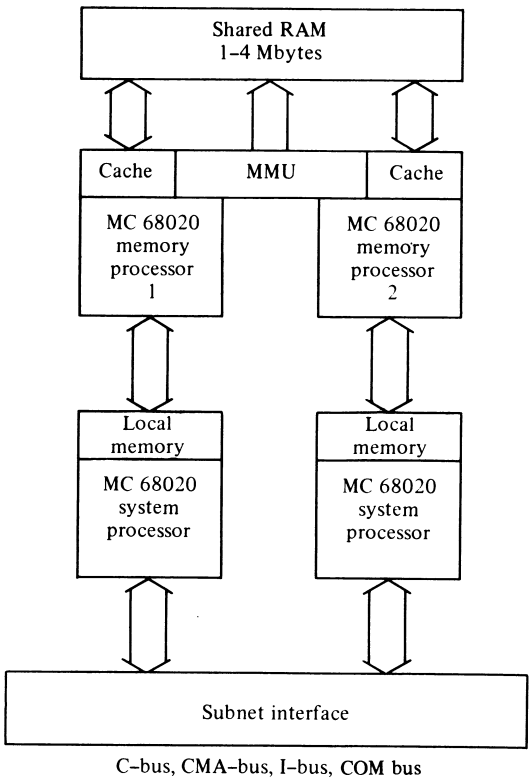
Shared RAM
1–4 Mbytes
Cache
MMU
Cache
MC 68020 memory processor 1
MC 68020 memory processor 2
Local memory
MC 68020 system processor
Local memory
MC 68020 system processor
Subnet interface
C–bus, CMA–bus, I–bus, COM bus

Figure 8. Design of proposed Phase II MCM (MCMII)

pre-fetch buffers together with another 8 Kbytes of fast dual-port memory, which can be used to pass data and commands between the two microprocessors. The slave also controls two National Semiconductor NS16081 Floating Point Units (FPUs), which are accessed as 16-bit peripherals and provide the arithmetic capability required by the Shell Model application. During Shell Model processing the slave handles all interaction with CMA-Bus and I-Bus as well as performing, with the aid of the FPUs, all arithmetic operations. As a guide, if MCMI performance is normalised to 1, then that of MCMII is approximately 9 during a matrix generating iteration in a Shell Model calculation.

On the basis of recent complete iterations on real nuclear data, the authors estimate that, with two MCMII modules in place, performance is approximately half that attainable on an IBM 360/195 mainframe using conventional Shell Model programming techniques.[4] Further, within the defined limits of the subnet, performance should increase almost linearly with the number of similar MCMs installed.

The Phase II MCM, now in the design stage, will be a powerful tightly coupled monoboard multiprocessor based on four MC68020 MPUs (Fig. 8), each equipped with a 'write-through' 8 Kbyte set-associative cache. In this design, the processors are paired, each pair consisting of a 'memory' processor with access to local bulk memory and a 'system' processor responsible for control of the subnet interface. The local bulk memory (1–4 Mbytes) is shared and divided into 1 Kbyte page-frames which may be dynamically designated cacheable or non-cacheable. When a task running on one of the processors attempts an access to shared memory the cache is checked while, concurrently, a local memory management unit (MMU) performs any address translations and checks access rights. If an access violation is detected the cycle is suspended or aborted; otherwise a request is issued to the on-board arbitration and a local shared-memory cycle is initiated. The MMU informs the cache whether or not the requested address falls in a cacheable page: if it does, the cache automatically stores the data as the processor reads it; if it does not, no such store may proceed. Thus only data in cacheable pages may be cached, avoiding the problem of cached data going 'stale' due to multiprocessor activity.

The proposed MMU will support demand-paged virtual memory and facilitate intertask protection in a much more general multiprogrammed multiprocessor environment. For the Shell Model application, the design of Fig. 8 is expected to yield a performance of approximately 30 on the above scale.

6. CONCLUSIONS

As outlined above, the MMPU designed for the Shell Model Processor project employs the latest 16/32-bit microprocessor technology to implement a small but powerful and flexible multiple CPU system. By emphasising modularity and linking the development to a particular microprocessor family, technological enhancement may be achieved without loss of user software compatibility. The MMPU global structures are designed to perform well above their currently projected load and it is hoped that, with scope for the integration of very-high-performance general-purpose processing elements and, indeed, of optimised dedicated processor modules where necessary, the range of applicability of the system will be significantly extended in the future.

Acknowledgements

The authors would like to thank Dr R. R. Whitehead of the Theoretical Nuclear Structure Group at Glasgow University for his assistance.

REFERENCES

1. E. T. Fathi and M. Krieger, Multiple microprocessor systems: what, why, and when. *IEEE Computer* (1983).
2. I. Barron, P. Cavill and D. May, Transputer does 10 or more MIPs even when not used in parallel. *Electronics* (17 Nov. 1983).
3. R. B. Pearson, J. L. Richardson and D. Toussaint, Special purpose processors in theoretical physics. *Communications of the ACM* **28** (4) (1985).
4. R. R. Whitehead, A. Watt, B. J. Cole and I. Morrison, *Computational Methods for Shell Model Calculations.* Advances in Nuclear Physics, vol. 9. Plenum Press, London (1977).
5. J. L. Baer, *Computer Systems Architecture.* Pitman, London (1980).
6. W. A. Wulf and C. G. Bell, C.mmp – A multi-miniprocessor. *AFIPS Conference Proceedings* 41, AFIPS Press (1972).
7. E. L. Dagless, M. D. Edwards and J. T. Proudfoot, The shared memories in the CYBA-M multi-microprocessor. *Proceedings of IEE,* E 301 (1983).

8. C. L. Seitz, The cosmic cube. *Communications of the ACM* **28** (1) (1985).
9. R. J. Swan, S. H. Fuller and D. P. Siewiorek, Cm* – A modular multi-microprocessor. *AFIPS Conference Proceedings* 46, AFIPS Press (1977).
10. P. Borrill and J. Theus, An advanced communications protocol for the proposed IEEE 896 Futurebus. *IEEE Micro* (1984).
11. Fastbus. A modular high-speed data acquisition system for high energy physics and other applications. US-NIM Committee DOE/ER-0189 (1982).
12. *VMEbus Specification Manual*, Rev. B. Motorola, Mostek, Signetics (1982).
13. *MC68020*, User's Manual. Motorola (1984).
14. E. Stritter and J. Gunter, A microprocessor architecture for a changing world: the Motorola 68000. *Computer* (1979).

Announcements

10–14 May 1987

APL 87, The International APL Conference on APL computer programming language, is to be held at the Fairmont Hotel, Dallas, Texas, USA. It is sponsored by the Special Interest Group of the Association of Computing Machinery and the Southwest APL Users' Group.

For further information please contact: APL 87 Registrar, 440 Northlake Shopping Center, suite 210, Dallas, TX 75238, U.S.A.

1–4 September 1987

13th International Conference on Very Large Data Bases, Brighton, England, U.K.

VLDB Conferences are a forum and focus for identifying and encouraging research, development, and the novel applications of database management systems and techniques. The Thirteenth VLDB Conference will bring together researchers and practitioners to exchange ideas and advance the subject. Papers of up to 5000 words in length and of high quality are invited on any aspect of the subject but particularly on the topics listed. All submitted papers will be read and carefully evaluated by the Programme Committee.

Programme

The programme will include an exhibition, six tutorials by eminent speakers which are specially oriented towards the needs of industry, and a high standard of refereed papers. The topics covered include: Data Models; Design Methods and Tools; Distributed Databases; Query Optimisation; Concurrency Control; Database Machines; Performance Issues; Security; Knowledge Base Representation; Multi-media Databases; Implementation Techniques; Object-Oriented Models; The role of logics.

Social Programme

There will be an extensive social programme including a civic reception, traditional English events, a conference dinner, sightseeing tours and 'weekend breaks' in London.

For further information and registration forms please contact:
Miss Christine Edginton, Conference Manager, BISL Conference Department, The British Computer Society, 13 Mansfield Street, London W1M 0BP (44-1-637 0471; Telex 262284).

7–11 September 1987

People and Computers HCI '87
The third annual conference of the BCS Human–Computer Interaction Specialist Group will be held at Exeter University, Devon, England from Tuesday 8 September to Friday 11 September 1987. The conference will be preceded by a day of tutorials on Monday 7 September.

The goals of the conference will again be: (i) to represent the current state of HCI, (ii) to increase communication between people working in the different disciplines of HCI and (iii) to discuss the future of HCI.

The conference has been planned in the knowledge that there is to be an international conference on a similar theme (Interact '87) in Germany the previous week. HCI '87 is designed to complement Interact '87. Many people who work in HCI in the U.K. will not be able to attend a conference held outside the U.K. Furthermore, the type of papers presented at the two conferences are likely to be of a different type. The papers in HCI '87 will be of a substantial length and will deal in detail with specific topics within HCI. In fact, HCI '87 plans to take advantage of the coincidence of Interact '87 by inviting to the U.K. international speakers, particularly from the U.S.A. and Japan, who will be in Europe at the beginning of September. There will also be workshops during HCI '87 that will report and discuss in detail issues raised, but perhaps not answered, during Interact '87. We hope that many of those who attend Interact '87 will also attend HCI '87 and play a major participatory role in making HCI '87 the success it has been in previous years.

For further details contact:

HCI '87 Conference, B.I.S.L., 13 Mansfield Street, London W1M 0BP. Telephone: (01) 637 0471.

8–11 September 1987

IFIP TC 8 Conference on Governmental and Municipal Information Systems will be held in Budapest, Hungary.

For further information please contact:

IFIP TC 8 Conference Secretariat, John von Neumann Society for Computing Sciences, Budapest 5, P.O.B. 240 H-1360, Hungary. Telephone: 361 329-390. Telex: 22 5369.