An Efficient Hardwired Router for a 3D-mesh Interconnection Network

Extended Abstract
J-L. Béchennec C. Germain, V. Neri.

LRI - Computer Architecture and VLSI Design
Bât 490 Université de Paris XI - Orsay
F-91405 Orsay cedex France
e-mail cecile@lri.lri.fr
fax 33 - 1 - 69 41 63 86

Abstract
We present a routing circuit intended to massively parallel message-passing architectures. This circuit meets the area requirements of a monochip Processing Elements and is time-efficient. It implements a new routing algorithm, Forced routing. We show that the forced routing algorithm may be hardwired in less than 10 mm² circuit in CMOS 1.2 µm process. Several design choices are discussed and tested for the arbiter, which is the most critical part of the circuit.

Keywords
VLSI, Routing algorithms, Hardwired routing, Highly parallel architecture

1. Introduction
We are currently studying message-passing massively parallel architectures, for special Artificial Intelligence applications (data and knowledge bases, semantic networks...). These applications need a huge number of small processes. Within a project called Mega (Machines to Explore Giant Architectures), we investigate the feasibility of machines with hundreds of thousands Processing Elements (PEs) to execute millions to billions of processes. Such machines are characterized by an asynchronous MIMD control, distributed local memories and message-passing communications. Intel Hypercube [1], Mosaic [2], the MDP [3] belong to this class of parallel machines.

Some novative features are necessary with such a number of PEs. As shown in [4], a 3-D grid is the most efficient interconnection network for such a number of processors. A hardwired router is necessary to implement efficient message-passing [5]. For the physical implementation of the machine, we use a fully modular approach, where each PE corresponds to a single VLSI chip packaged in a special MegaPack. This packaging has been defined for a direct 3D physical and electrical interconnection, without boards or racks [6]. The choice of a monochip PE implies that the main constraint on the design will be the chip density of currently used VLSI CMOS technologies. The total chip area, which cannot exceed 1 cm² for an acceptable yield, must be used for the CPU, the routing circuit, and the local memory. Even if some specific dynamic RAM cell compatible with a standard CMOS process [7] is used, the local RAM uses most of the chip area (near 80%).

Figure 1 presents the Mega PE design. The CPU has a Risc architecture, devoted to concurrent execution of functional processes. The Risc approach is first demanded by the very small area available, but fits very well with the on-chip memory that always can act as a cache. A complete description of the CPU can be found in [8].

In this paper, we present the design of the routing circuit, the Domain Management Unit (DMU). Routing performance, both the average message delay and the network throughput, is decisive for a massively parallel architecture [9]. In the second part of this paper, we present a new routing algorithm, Forced routing. It is a tradeoff between the classical greedy algorithms, which are quite easy to hardwire [10], but behave poorly at high network load, and the randomized algorithms [11], which are asymptotically optimal but area-consuming. The third part describes the DMU architecture and shows that it meets the area requirements. Part three and four discuss the design of the
DMU heart, the arbitration circuit, both from logical and electrical points of view.

2. The Forced routing algorithm

Routing means both switching techniques and routing algorithms. Wormhole routing [121], is an appealing technique in our case, because it pipelines the flits (atoms of transmission) with low buffering capacity. However, Wormhole routing is almost ever associated with deterministic algorithms, for deadlock-avoidance purpose [13]. This leads to unnecessary contention, thus lowers performance: from source to destination, only one path will be used. So we adapted the fundamental idea of Valiant’s algorithms, that is to reduce contention by randomization, to our asynchronous context. Forced Routing may be described by the following algorithm:

```
for input_link = 0 to network_degree do
  if there exists a message on input_link then
    if shortest path output links are free then
      randomly allocate one
    else
      if output links are free then
        randomly allocate one
      else
        buffer message
    endif
  endif
end for
```

It is adaptive as it insures a large, even though not optimal, occupation of the network channels; it saves chip area by buffering the messages that cannot progress to their destination on the network, and not on on-chip buffers. It is forced in the sense that, even when there are conflicts between incoming messages, all of them are routed, possibly along directions moving them away from their destination. So the path length, and thus the routing delay, will gracefully degrade with contention. Deadlock is not possible, because messages never stop moving in the network, and starvation is avoided by giving the highest priority to the internal channel, which serves the routing buffer. A complete description of the Forced routing algorithm, with extensive performance simulations, can be found in [4].

3. The Domain Management Unit architecture

3.1. General organization

Figure 2 shows the DMU architecture. There are seven input channels, one-byte wide. Six external channels input the external messages from the 3D-grid interconnection network; one internal channel is multiplexed between the routing buffer and the processor (for emission). Each channel is associated with a Channel Controller. Each controller includes three one-byte registers to buffer the message destination address, a logic circuit to generate the requests on output channels, and a small PLA for error detection purpose. The complete address must actually be known before routing. This address is in relative format (Δx, Δy, Δz) rather than absolute: this format saves time and area, because the sign bit may be directly used.

The Arbitration Circuit drives a Crossbar, which connects the input channels to the output channels. When passing through the crossbar, the address header is adjusted following the chosen output channel. The Routing Buffer is a hardwired FIFO: emission and reception may be pipelined. The Ejector stores in the local memory the messages that have reached their final destination. Ejection is not always possible: only one message can be received at a time, the CPU may be in critical section for input/output or the memory stack may be full. In this case, the message will be derouted and will come back later. When ejection is accepted, the DMU inhibits the CPU clock, so that the ejection is transparent to the CPU.

The most important part of the DMU is the arbitration circuit. It implements in hardware the Forced routing algorithm, and the whole network throughput is determined by its performance: no control means on the messages flow are available, so the arbitration circuit must work at the same frequency as the network physical links. First speed evaluations have shown that, for standard CMOS circuitry without optimization, the propagation time for the arbiter would be 50 ns, and 50% of the total DMU critical path.

3.2. Surface evaluation

Here we consider the 1.2 μm CMOS process of SGS-Thomson Corp, and a full-custom layout. Table 1 summarizes
the surface evaluations for the different parts of the DMU, based on the actual layout of the complex operators. We assume that wiring is 50% of the total area. The reasons for this unusual overhead are twice: first the number of parallel wires is very high (the data path from the input channels to the crossbar is 56 bits wide); moreover, the DMU must be connected to a large number of I/O pads, located on the borders of the IC. The Routing Buffer was separately evaluated. It includes a 10-bit 64-word dual-ported static RAM, and some extra logic to manage it as a circular buffer. This circuit is very repetitive, thus wiring consumes only a small part of its area.

4. The Arbitration circuit design

The arbitration circuit has two inputs: the requests from the input channels, i.e. the output channels that may be used to build a shortest path, and the current crossbar state, i.e. the output channels occupancy. It allocates the available channels following the Forced routing algorithm.

Three designs may be used for the arbiter: a fully parallel scheme, polling or a priority circuit. The first solution is the only one that can compute the optimal input-to-output permutation. Here, optimal means that this scheme achieves the minimal distance between the (eventually conflicting) requests and the actual permutation. Unfortunately, this solution would imply an associative memory with 5040 (fact (7)) cells, thus a prohibitive silicon area. Polling may be implemented in a compact way by automatas, but implies high-frequency clocks. The solution we choose is a sequential structure (Figure 3), with one stage by input channel. Thus if simultaneous arrivals of messages happen, the input channels are served in a fixed-priority order. Stage 0 is devoted to the internal channel, so that it has the highest priority, as required by the Forced routing algorithm.

Each stage consists of four parts (Figure 4). The request generator gets the requests from the corresponding input channel and the crossbar state; it generates a request on all the output channels that can constitute a Forced routing path. The stage-arbiter arbitrates between these requests by coming up to the first one in a predefined order. The request generator may be described by the following equations (all input channels indexes have been omitted for clarity): let \( r_j \) be the variable associated with a request on output channel \( j \) and \( o_j \) denote the \( j \) channel state (\( o_j = 1 \) if channel \( j \) is requested and \( o_j = 1 \) if channel \( j \) is free); the acceptable requests \( d_j \) are:

\[
d_j = o_j \cdot r_j
\]

When none of the requested channels are available, all free output channels may be allocated to the message. The \( D \) function indicates this situation:

\[
D = (o_0 \cdot r_0, o_1 \cdot r_1, o_2 \cdot r_2, o_3 \cdot r_3, o_4 \cdot r_4, o_5 \cdot r_5)
\]

Then the new crossbar state, \( s_j \), is computed and carried out to the next stage of the arbitration circuit. When there is no actual request on the current stage, either if the channel is currently forwarding the data part of the message or if there is no message at all, the input crossbar state must be the copied out as output. The \( q \) signal, generated by the input controller (\( q = 1 \) for an active message), is used to multiplex the two cases:

\[
s_j = q \cdot s_j + \bar{q} \cdot s_0
\]

Because of the fixed priority order in the stage-arbiter, the previous arbitration circuit will not satisfy the Forced routing specifications. Thus a shifter creates a random circular permutation of the requests. The reverse shift at the arbiter output ensures the correct crossbar command.

A most straightforward implementation of the randomization used in the Forced routing algorithm would be to randomly choose the actual output channel between the request generator outputs, at each stage. However, this solution presents time and area drawbacks. The extra hardware is on the arbiter critical path, and the random generators must be replicated on each stage. Moreover, the extent of the random choice depends on the number of available output channels, so the random generators needs the request generator outputs. Thus the random generators are on the critical path too. The first solution is advantageous from the area criterion, because it needs a single random number generator, and because the crossbar state will have to be
shifted only once (Figure 5). It saves times too, because the shifter may be external to the arbitration circuit, thus contributes only for a minor part to the critical path.

5. The arbitration circuit electrical design

All speed evaluations use the electrical simulator ST-Spice, with the MOS transistors typical models for the 1.2 μm process of the SGS-Thomson Corp.

5.1 The request generator

The simplest way to implement the D function is the complex operator shown by Figure 7. The best circuitry style for this operator seems to be a pseudo-nMOS gate (Figure 6).

Its drawback is high power consumption. However the penalty for the whole circuit will not be too severe, because all other functions use conventional CMOS logic.

The simulated propagation delays are, with extracted capacitance load and typical simulation case (27°C, 5V):

- \( t_{phl} = 0.45 \text{ ns} \)
- \( t_{plh} = 0.85 \text{ ns} \)

Complex gates are used for the (dj) computation too, but in conventional CMOS (figure 8). The CMOS regenerates the signal coming from the D operator, which has a bad low level.

The propagation delays are

- \( t_{phl} = 0.2 \text{ ns} \)
- \( t_{plh} = 0.3 \text{ ns} \)

5.2 The stage-arbiter

The stage-arbiter may be designed in many ways. At the logical level, it may be designed either in a standard style, where each \( c_j \) is separately computed, or with auxiliary functions, such as the Gi and Pi (Generate & Propagate) used for the carry look-ahead in adders.

Figure 9 presents the last solution. The idea is to compute two functions \( x_j \) and \( y_j \) along a chain of transistors. Table 2 shows the logical values of \( x_j \) and \( y_j \) for an equivalent operator limited to three entries. The logical and of the two functions gives the \( c_j \). The inverters regenerate the propagating signal and imply a complemented logic along the chain.

This solution somehow degrades the performance towards the best one (Table 3). However, if the decisive criterion is transistor number and design simplicity, it remains
realize Nands with from two to six entries. If power dissipation is not critical, pseudo-nMOS Nors may be used precharged dynamic circuitry. This solution is not very appealing that will make the circuit and the layout more standard CMOS circuitry (i.e. c3, c4, c5 and cg). The last way is lor the functions which are penalized by too many entries in

In a standard solution, three logics may be used to compute the cj. We can first use CMOS standard gates to realize Nands with from two to six entries. If power dissipation is not critical, pseudo-nMOS Nors may be used for the functions which are penalized by too many entries in standard CMOS circuitry (i.e. c3, c4, c5 and c6). The last way is precharged dynamic circuitry. This solution is not very appealing: the precharged circuitry implies a clock management that will make the circuit and the layout more

<table>
<thead>
<tr>
<th></th>
<th>CMOS</th>
<th>pseudo-nMOS</th>
<th>propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical / 5V / 27° C</td>
<td>2.9 ns</td>
<td>1.7 ns</td>
<td>3.6 ns</td>
</tr>
<tr>
<td>Typical / 5V / 117° C</td>
<td>4.0 ns</td>
<td>2.4 ns</td>
<td>5.0 ns</td>
</tr>
<tr>
<td>Typical / 4.5 V / 27° C</td>
<td>3.4 ns</td>
<td>2.0 ns</td>
<td>4.7 ns</td>
</tr>
<tr>
<td>Slow / 5V / 27° C</td>
<td>3.6 ns</td>
<td>2.1 ns</td>
<td>4.9 ns</td>
</tr>
</tbody>
</table>

Table 3: Performance comparison for the stage-arbiter

<table>
<thead>
<tr>
<th></th>
<th>CMOS</th>
<th>pseudo-nMOS</th>
<th>propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical / 5V / 27° C</td>
<td>4.5 ns</td>
<td>3.3 ns</td>
<td>5.2 ns</td>
</tr>
<tr>
<td>Typical / 5V / 117° C</td>
<td>6.1 ns</td>
<td>4.5 ns</td>
<td>7.1 ns</td>
</tr>
<tr>
<td>Typical / 4.5 V / 27° C</td>
<td>5.2 ns</td>
<td>3.8 ns</td>
<td>6.5 ns</td>
</tr>
<tr>
<td>Slow / 5V / 27° C</td>
<td>5.5 ns</td>
<td>4.0 ns</td>
<td>6.8 ns</td>
</tr>
</tbody>
</table>

Table 4: Propagation time for a single stage

intricated. Moreover, the inputs to dynamic operators cannot present transients that may lead to a parasitic discharge of the tree before the exact inputs are established. Last, dynamic values may be quite difficult to test.

The performance comparisons summarized in Table 3 shows that the pseudo-nMOS solution is the best.

A simple multiplexer, with two pass-transistors generates the sj outputs. Table 4 presents the performance of a whole stage of the arbitration circuit.

5.3 : The Shifter

The shifting function is made of parallel multiplexors, designed with pass-transistors. Setting one A1 line to 1 and the others to 0 results in shifting the entries from 0 to 5 positions.

Assuming a pseudo-nMOS design, the propagation time for the arbitration circuit is near 22 ns. Including the possible optimizations on the DMU other parts, a 60 ns cycle becomes possible. The contribution of the arbitration circuit to the total propagation time is now only 41%. This performance leads to a 16 Megabyte/s link bandwidth.

Conclusion

Efficiency of massively parallel architectures widely depends on their communication design. The routing algorithm must make intensive use of the interconnexion network, which is the rarest resource of the machine. However, the monochip design of the MegaPE severely limits the routing algorithm by area constraints too. The forced routing algorithm achieves an efficient tradeoff between these two objectives. It avoids contention by randomly spreading the messages on the same length paths, and it may be hardwired in a small area (less than 7 mm2) with a time performance fully compatible with the global PE design.

Acknowledgements

The authors wish to thank Professor D. Etiemble and Research Director J-P Sansonnet for their useful suggestions and discussions.

References

[8] F. Capello, J-L Béchennec, D. Etiemble, "A RISC Central processing Unit for a Massively Parallel Architecture", EUROMICRO 90, Amsterdam, August 90