Exploring Optimal Cost-Performance Designs for Raw Microprocessors

Csaba Andras Moritz   Donald Yeung   Anant Agarwal
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
{andras,piano,agarwal}@lcs.mit.edu

Abstract

The semiconductor industry roadmap projects that advances in VLSI technology will permit more than one billion transistors on a chip by the year 2010. The MIT Raw microprocessor is a proposed architecture that strives to exploit these chip-level resources by implementing thousands of tiles, each comprising a processing element and a small amount of memory, coupled by a static two-dimensional interconnect. A compiler partitions fine-grain instruction-level parallelism across the tiles and statically schedules inter-tile communication over the interconnect. Because Raw microprocessors fully expose their internal hardware structure to the software, they can be viewed as a gigantic FPGA with coarse-grained tiles, in which software orchestrates communication over static interconnections.

One open challenge in Raw architectures is to determine their optimal grain size and balance. The grain size is the area of each tile, and the balance is the proportion of area in each tile devoted to memory, processing, communication, and I/O. If the total chip area is fixed, more area devoted to processing will result in a higher processing power per node, but will lead to a fewer number of tiles.

This paper presents an analytical framework using which designers can reason about the design space of Raw microprocessors. Based on an architectural model and a VLSI cost analysis, the framework computes the performance of applications, and uses an optimization process to identify designs that will execute these applications most cost-effectively.

Although the optimal machine configurations obtained vary for different applications, problem sizes and budgets, the general trends for various applications are similar. Accordingly, for the applications studied, assuming an 1 billion logic transistor equivalent area, we recommend building a Raw chip with approximately 1000 tiles, 30 words/cycle global I/O, 20Kbytes of local memory per node, 3-4 words/cycle local communication bandwidth, and single-issue processors. This configuration will give performance near the global optimum for most applications.

1 Introduction

Advances in semiconductor technology have made possible the integration of multiple functional units, large cache memories, reconfigurable logic arrays and peripheral functions into single-chip microprocessors. Unfortunately, increases in the performance of contemporary microprocessors have come at the cost of increasing inefficiencies in silicon area usage. The inefficiencies arise from the complexity of designs that use hardware support to exploit more instruction level parallelism.

Maintaining a rapid increase in microprocessor performance will require a cost efficient utilization of silicon area. The MIT Raw microprocessor is a proposed architecture that exposes its internal hardware structure to the compiler, so that the compiler can determine and orchestrate the best mapping of an application to the hardware. A Raw microprocessor [1] is reminiscent of a coarse-grained FPGA and comprises a replicated set of tiles coupled together by a set of compiler orchestrated, pipelined, switches (Figure 1). Each tile contains a simple RISC-like processing core and an SRAM memory for instructions and data. Instruction memory allows the multiplexing of the compute logic on a cycle by cycle basis. SRAM memory distributed across the tiles eliminates the memory bandwidth bottleneck, provides low latency to each memory module, and prevents off-chip I/O latency from limiting effective computational throughput.

The tiles are interconnected by a high-speed 2D mesh network, allowing inter-tile communications to occur with register-like latencies. The switches themselves contain some amount of SRAM so that the compiler can load into the switch a program that multiplexes the interconnect in a cycle by cycle fashion, just as in a virtual-wires based multi-FPGA system [4].

A typical Raw system includes a Raw microprocessor coupled with off-chip RDRAM (RamBus DRAM) through multiple high bandwidth paths. The two level memory hierarchy, namely, a local SRAM memory attached to each tile inside the Raw chip, and a large external RDRAM memory, is necessary to be able to solve large problems that exceed the size of the on-chip memory.

Raw architectures achieve the performance of FPGA-based custom computing engines by exploiting fine-grained parallelism and fast static communication, and by exposing the low-level hardware details to facilitate compiler orchestration. Unlike FPGA systems, however, Raw machines support instruction sequencing and are more flexible because the execution of a new operation can be accomplished merely by pointing to a new instruction. Compilation in Raw is faster than in FPGA systems because it binds into hardware commonly used compute mechanisms such as ALUs and memory paths, thereby eliminating repeated low-level compilations of these macro units. Binding of common mechanisms into hardware also yields better execution speed, lower area, and better power efficiency than FPGA systems.

The designer of an FPGA device or a Raw microprocessor is faced with the challenge of determining the best division of VLSI resources among computing, memory, and communication. This challenge is termed the *balance problem*. Furthermore the designers of both an FPGA and a Raw device must address the *grain size* issue – in other words, whether to implement a few powerful tiles, or whether to use many small tiles each with a lower performance.

This paper presents an analytical framework with which designers can reason about the division of resources in a VLSI chip. Although our analysis in this paper is focussed on the Raw microprocessor, the analysis generalizes to other architectures. Our objective in this paper is to gain more insight into cost-performance optimal designs given a fixed amount of resources.

The framework presented in this paper focuses on the performance requirements of applications, introduces an *architecture model*, a *cost model* and a *performance model* for applications, and defines an optimization process to search for performance optimal designs given a cost constraint.

The architecture model defines an architecture based on parameters that include the number of tiles P, the processing power of each tile p, the amount of memory in each tile m, and the communication bandwidth out of each tile c. The cost model estimates the cost in terms of chip area of realizing the given architecture with the specified set of parameters. The performance model estimates the runtime of each application as a function of the problem size. Performance estimation is based on both (1) a characterization of the application and its algorithms in terms of its requirements including processing steps, memory and communication volumes, and (2) the architecture model.

Together with a cost constraint defined in terms of the cost model, our performance model allows us to perform a constrained optimization on the independent architectural variables. We can, for example, compute the points or contours in the architectural space that correspond to the best performance for a given cost, lowest cost for a given level of performance, or best efficiency defined by performance/cost.

The algorithms used in this study have been adapted to the Raw system architecture illustrated in Figure 1 by first partitioning them into subproblems that can fit within the Raw chip. Each subproblem is loaded from the external global RDRAM memory into the set of local memories in the tiles. Computation occurs on the subproblem, and the results are stored back into external RDRAM. All the subproblems are visited (possibly multiple times) in sequence. The algorithmic slowdown due to blocking the problem in this manner is accurately modeled. Each subproblem is solved in parallel with a blocking algorithm. Applications studied in this paper include Jacobi Relaxation, Dense Matrix Multiply, Nbody, FFT, and Largest Common Subsequence.

The specific contribution of this paper include:

- A general framework for reasoning about the design space of VLSI-based parallel architectures including models for cost and performance.
- Insights on optimal grain size and balance in Raw microprocessors.

The remainder of this paper is organized as follows. Section 2 describes the three models developed in this paper: the performance model, the cost model and the application model and gives a qualitative analysis of cost and performance. Section 2.7 formulates the optimization process based on previous model assump-

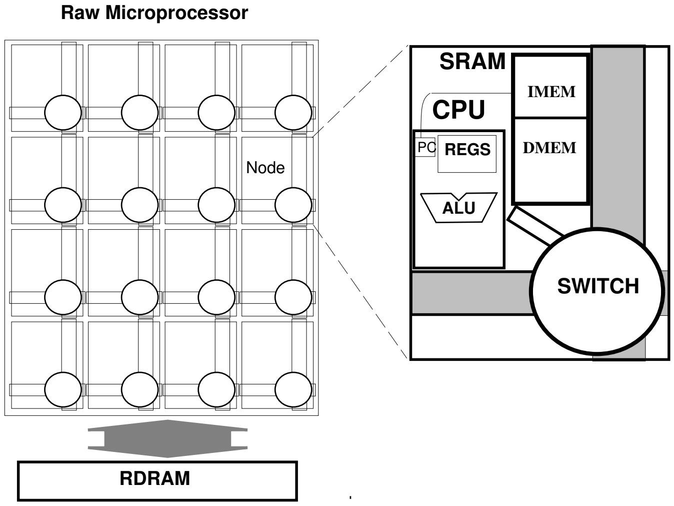

Figure 1: Raw system composition. A typical Raw system includes a Raw microprocessor coupled with off-chip DRAM and stream IO devices. Each Raw tile contains a simple RISC-like processor, an SRAM memory for instructions and data, and a switch. The tiles are interconnected in a 2D mesh network that is orchestrated by the compiler. The switches themselves contain some amount of SRAM so that the compiler can load into the switch a program that multiplexes the interconnect in a cycle by cycle fashion, just as in a virtual-wires based multi-FPGA system.

tions, and Section 3 gives our experimental results. Section 4 concludes the paper.

2 Framework

This section presents the analytical framework used in analyzing candidate designs in terms of their grain size and balance. We first start with a motivation for a study of grain size issues.

2.1 Motivation

Two key questions in the design of a Raw microprocessor involve the *grain size* of its tiles and their *balance*. The grain size reflects the sizes of various components inside the tiles such as memory, processing, and communication. A very coarse grain design would involve multiple-issue superscalars for processing and large local memories. Very fine grain designs would be similar to contemporary FPGAs and include a few bits worth of logic and memory within each tile, and a few wires connecting the individual tiles. Designs with a moderate grain size would involve very simple single-issue processors in each node.

Grain size and balance play a large part in determining the efficiency or performance per unit cost of a machine assuming a fixed total budget. If an engineer builds a small number of very large (coarse grain) nodes, a point of diminishing returns is reached where node performance increases very slowly (if at all) as node size is increased. On the other hand, building a large number of very small (fine grain) nodes will also result in diminishing returns as the communication costs dominate. The highest efficiency occurs at an optimal point between the two extremes. Similarly, as observed by Kung and others [11, 5], there is an optimal balance of resources between the processor, memory, and communication components within a node.

While there has been much debate on this topic, few concrete results have been reported. Machine balance and grain size continues to be determined more by convenience and market forces than by engineering analysis. Our primary motivation in undertaking this study is to provide an analytical framework to enable engineers to obtain insights into the tradeoffs in choosing various machine parameters.

Let us first provide an overview of the framework. Table 1 summarizes our notation organized by model category. Throughout the paper, execution times are measured in *machine cycles*, information in units of machine *words*, and cost in *SRAM bit equivalents (Sbe)*. As discussed in Section2.4, an Sbe is the area

ARCHITECTURE MODEL	
P	number of tiles in Raw
p	processing power of each tile
m	amount of SRAM in a tile
c	local communication bandwidth
l	single hop interconnect latency of a word
o	software overhead for communication
k_d	average network distance traversed by messages
l_g	DRAM latency for global communication
b_g	global communication bandwidth or DRAM accessing bandwidth
x	machine configuration: $x = < P, p, m, c, l, o, k_d, l_g, b_g >$
COST MODEL	
$K_p(p)$	processor cost per tile
$K_m(m)$	memory cost per tile
$K_c(c)$	local communication router cost per tile
$K_{lg}(l_g)$	global latency cost for entire Raw system
$K_{bg}(b_g)$	global bandwidth cost for entire Raw system
$K(x)$	total cost of Raw system
APPLICATION MODEL	
N	problem size
N'	subproblem size: part of the original problem requiring one global step
$R_p(P, N, N')$	total amount of computation required per tile to solve the problem
$R_m(P, N, N')$	total amount of local memory required per tile
$R_m^a(P, N, N')$	total amount of local memory per tile required to hold a subproblem
$R_m^l(P, N, N')$	total amount of buffer per tile for overlapping local communication
$R_m(P, N, N')$	total amount of buffer per tile for overlapping global communication
$R_c(P, N, N')$	total amount of data required to be sent or received in-between tiles
$R_o(P, N, N')$	software overhead attached to local communication events
$R_{bg}(P, N, N')$	total amount of global communication required per chip
$R_{lg}(P, N, N')$	total amount of overhead and latency required for global communication
PERFORMANCE FUNCTIONS	
$T_s(N, p)$	sequential run-time of application
$T(N, N', x)$	parallel run-time of application
$T_p(N, N', x)$	computation time per node including overheads
$T_c(N, N', x)$	local communication time per node
$T_g(N, N', x)$	global communication time per node
$T_{ai}(N, N', x)$	amount of overhead per tile resulting from algorithmic load imbalance

Table 1: Overview of model parameters and functions.

occupied by one bit of SRAM memory.

2.2 Overview of the Framework

Let us overview our analytical framework illustrated in Figure 2 by considering a simple machine model. In its simplest form, a parallel machine can be characterized by the number of tiles or nodes P, the processing power of each node p (operations per cycle), communication bandwidth of each node c (words per cycle), and the amount of local memory per node m (words).

For a given problem size and partitioning strategy, an application can be described by its processing, communication and memory requirements, or R_p (operations to be performed), R_c (words to be communicated) and R_m (words).

The performance of the application in terms of its runtime T is derived from the application requirements and the architectural model. If the processing time $T_p = \frac{R_p}{p}$ and the communication time $T_c = \frac{R_c}{c}$, then if processing and communication is fully overlapped, the runtime is given by $T = max(T_p, T_c)$.

We use cost models $K_p(p)$, $K_c(c)$, $K_m(m)$ to map the machine parameters P, p, c, m into costs. In other words the processor cost model $K_p(p)$ provides the area cost of implementing a processor that can perform p operations per cycle. The total machine cost for a P processor machine is then $K = P(K_p + K_c + K_m)$.

Given an application with a fixed problem size N and an area budget B, a constrained optimization problem is defined with the objective of finding the optimal machine configuration that gives the smallest runtime for that budget. In other words the framework finds the set of architectural parameters P, p, c, m

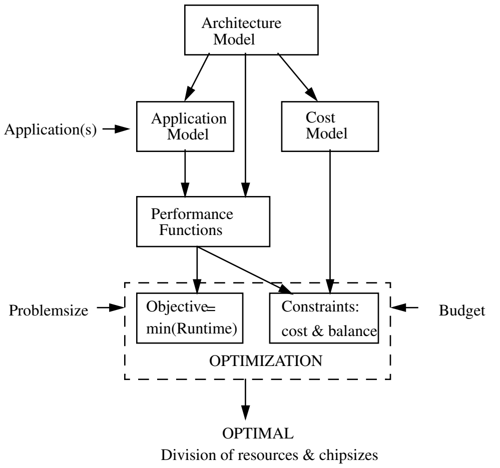

Figure 2: Analytical framework. The key components of the framework are the models and the optimization process. Given an application with an associated problem size and a fixed budget, the constraint equations are derived for the optimization. The nonlinear optimization process searches the machine configuration space that gives the minimal runtime for the application.

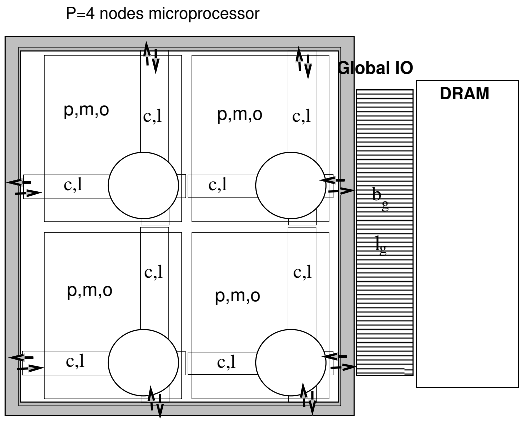

Figure 3: A four node illustrative Raw system characterized by the parameters $< P, p, m, c, l, o, b_g, l_g >$ where the processing power per node in operations per cycle is p, the amount of SRAM memory per node is m, the local communication bandwidth per node in words per cycle is c, the software overhead for a communication event in cycles is o, the single hop communication latency is l, the global off-chip communication bandwidth per Raw chip in words per cycle is b_g, the RDRAM latency expressed in cycles is l_g.

that yield a minimum value for T, given that the cost K cannot exceed the available budget B. Or more formally,

find P, p, c, m
to minimize $T = max(T_p, T_c)$,
subject to $B \geq K$.

As discussed in more detail later, the optimization process is sped up by a set of *balance constraints.* The balance constraints state that for the optimal solution the computation time and communication time must be equal, and that the physical memory should fit the problem. The balance constraints greatly reduce the size of the search space, and thus the complexity of the optimization procedure.

The following sections discuss each of the components of the framework and the optimization process in more detail.

2.3 Architecture model

This section discusses parameters necessary for architecture characterization. Although several approaches to modeling the performance of a parallel computer have been proposed in the literature [2, 3], none are completely suited to modeling fine-grain parallel systems built on a chip. Figure 3 shows our characterization of a Raw system using the parameters described below. Our machine characterization differs from previous ones in the sense that it captures both local and global communication performance, and includes software overheads.

We choose as independent parameters the number of nodes, P, the processing power per node in operations per cycle, p, the memory per node m in words, the local communication bandwidth per node in words per cycle, c, the software overhead for communication in cycles, o, the single hop latency of the network, l, the global off-chip communication bandwidth per node in words per cycle , b_g, and the RDRAM latency expressed in cycles, l_g.

As an example, sending a local inter-tile message of length L words first involves spending o cycles in launching the message. The message header word travels on average a distance of k_d hops in the network, using l cycles per hop. Because the bandwidth out of a node is c words per cycle, subsequent message words take $\frac{1}{c}$ to enter the network. The receiving tile would also spend o cycles receiving the message. Thus, the communication time per message is

$$T_c = 2o + k_d l + (L-1)\frac{1}{c} \quad (1)$$

Writing a block of data to the off-chip RDRAM memory first involves an overhead o associated with starting up global communication. The latency of accessing the DRAM will be the sum of the latency of traversing the interconnection network in one dimension ($k_d l/2$) plus l_g the DRAM latency. (We divide by two to indicate that RDRAM memory messages do not have to traverse both the X and Y network dimensions) The transfer rate of subsequent words will be the minimum of the local communication bandwidth and the global communication bandwidth per tile (since multiple tiles might be writing external memory). Thus the time for a writing a block of size L to memory is,

$$T_g = o + \frac{k_d}{2} l + l_g + (L-1) max \left(\frac{1}{c}, \frac{P}{b_g} \right) \quad (2)$$

Communication locality can be captured at the application level by accounting for it in the average distance that messages travel (k_d). We ignore contention effects (e.g. resource and network contention) also because we assume that the compiler can statically orchestrate communication events much as in a virtual-wires system. We also use a conservative approach in defining applications' communication requirements.

2.4 Cost model

We use silicon area as a measure of cost. Silicon area reflects the fundamental cost of building a component and is a good basis for comparing alternatives as opposed to market price which includes many artificial factors. The cost model is based on CMOS microprocessors, SRAM and DRAM memories, and a mesh interconnection technology. For simplicity we consider the off-chip RDRAM memory free. Although our assumptions may change specific numerical results, the methodology for determining balance and grain size remains the same.

We normalize cost to units of SRAM bits, viz. one bit of SRAM takes one unit of area and therefore one unit of cost. We express the cost of all other components in terms of *SRAM bit equivalents* (Sbe).

We use the notion of relative density to enable the normalization of logic, memory and communication areas into units of SRAM bit equivalents. Relative density captures the area impact of wires and more irregular structures such as logic areas versus the more regular memory arrays. Although an SRAM bit comprises typically 4 to 6 transistors we observe that the area it occupies is similar to the area of a logic transistor in a CPU die because of its regular structure and therefore its higher relative density. Thus, the chip size expressed in Sbe units is equivalent to the total number of transistors for logic areas.

Area	Relative density
CPU logic transistor	1 Sbe
Router logic transistor	1 Sbe
SRAM bit	1 Sbe
DRAM bit	1/16 Sbe

Table 2: Relative densities of constituent VLSI components.

A DRAM bit is realized with one transistor and the area it occupies is 10-16 times smaller than an SRAM bit area. We arrived at this conclusion as the typical SRAM cell requires a wire grid of dimension 3×4 compared to a DRAM cell implemented on the intersection of two wires. Factors such as the number of metallic layers may change the relative density relations as more layers increase the density of logic areas. The logic area density is also reduced because of the greater amount of area devoted to wiring.

The following cost functions are based on empirical observations and statistics gathered on current implementations of superscalars and router chips.

Processor cost K_p The processor cost model computes the area cost as a function of p. We find it convenient to relate p to cost k_p using an intermediate parameter i, which is the number of issue units i in the processor. Thus, $i = 4$ implies a 4-Way superscalar with a maximum of 4 operations per cycle.

We model the relationship between processing cost and instruction issue structure as a quadratic curve, which captures the cost increase due to multiple issue superscalars.

$$K_p(i) = B_p + K_{ps}(i-1)^2 \quad (3)$$

In the above, a cost of B_p is required to achieve a single issue processor with $i = 1$.

We relate processing power p and the number of issue units i using:

$$p = \sqrt{i} \quad (4)$$

This model captures the relationship between performance and cost due to more aggressive clock rates of lower issue processors. Typically single issue designs obtain 1.6-2 times faster clock rates than corresponding high-issue rate processors. It also captures the fact that it is easier to obtain performance close to the theoretical maximum cycles per instruction in lower-issue processors as they require a smaller amount of instruction-level parallelism in applications.

Studying the layout of some simple RISC processors [6, 14, 13, 8] leads to a base cost of $B_p = 2.5 \times 10^5$ transistor. That is, a minimal single issue 64 bit processor can be built in the area of 250K SRAM bits or with 250K logic transistors. A cost constant of $K_{ps} = 4 \times 10^5$ Sbe, was arrived at from the study of some high-end processors [22, 20, 21, 8].

For validation, Figure 4 compares the number of transistors dedicated to logic in several superscalar microprocessors with our cost model for $K_p(i)$. We observe that for higher-issue superscalars the variation in the number of transistors dedicated to logic areas is large. This variation is caused by important differences in implementation of components like issue structure, scheduling and memory interfaces. A more detailed cost model for superscalars may also deal with the cost impact of dynamic or static issue structures, scheduling and memory interfacing.

Memory cost. We approximate memory cost as a linear function of capacity m.

$$K_m(m) = B_m + Wm \qquad (5)$$

Here, m is the memory size in words, K_{ms} is the cost per word of memory, and B_m is the fixed overhead cost of the memory. This overhead includes logic for translation, address decode, data multiplexing, and memory peripheral circuitry. For our calculations, we assume that $W(wordsize) = 64$, and the overhead, B_m, is 5×10^4.

Communication cost. Main components of a typical router comprise a routing module, a crossbar arbiter, and input output modules often including large FIFOs. We observe that most of the area in current router chips are taken up by FIFOs and pad frames (c.a. 20%). The amount of FIFOs depends on such factors as the number of virtual channels. The area of queues reflects size of message flits and a length which is typically 16-20 flits. A flit is the number of bits transferred in one cycle, and therefore it also equals c expressed in bits. One word per cycle communication bandwidth thus requires a flit size of one word. Although not necessary, we also assume the flitsize is equal to the physical channel width. We denote the dimension of the network as n. The total number of bidirectional channels is then $2n$. Our results focus on two-dimensional networks, so $n = 2$ for most of this paper. Logic areas such as the crossbar usually occupies a small part of the total area. The cost function for the routers is described in the following equation:

$$K_c(c) = B_c + K_{cs} W \times FIFOl \times 2n \times SetofQ \times c. \qquad (6)$$

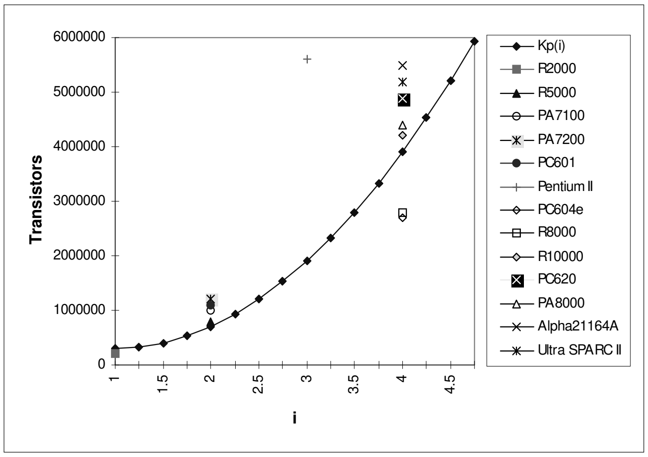

Figure 4: Comparison between the processor cost function $K_p(i)$ and cost of logic areas in current superscalar microprocessors. Cache memory areas are factored out.

In the equation above, $FIFOl$ is the length of the FIFOs and $SetofQ$ is the number of queue sets due to virtual channels. Our results use $SetofQ = 1$. The communication cost factor, K_{cs}, is derived by fitting the cost function equation with the areas of routing chips shown in Table 3.

For our calculations, we use $K_{cs} = 25$. For example, a router with a 64 bit flit size and with one set of queues, each with length 16 flits, takes approximately 125000 logic transistor area in our model.

The base area for a router, B_c is estimated at 2.5×10^4. We arrive at this from a study of simple routers [10, 9, 6, 15]. Examples of routers with the number of transistors used in current implementations are shown in Table 3. The The estimates using our communication cost model are also shown. The comparison indicates that our cost model reflects relatively accurately the area occupied by these routers except the RDT [7] router chip that has more than half of its area devoted to a multicast mechanism module and a bit-map generator.

Global communication cost. We approximate global communication cost as a linear function of global off-chip communication capacity. The base area for global I/O, $B_{bg} = 10^4$, is estimated to be somewhat smaller than a simple router area as no routing functions are necessary. The global communication bandwidth is limited by the maximum number of pins a packaging technology will allow. As current microprocessor packaging technologies use from one hun-

Router	Transistors	Estimated	Type	Network	Flits	FIFOl	SetofQ	Pins
J machine	29000	29050	wormhole	3D mesh	9	3	1	-
Postech	30140	31400	virtual cut-through	2D mesh	8	8	1	100
Mosaic C	60000	44200	wormhole, asynch.	2D mesh	8	4	3	c.a. 88
Chaos	110000	121000	chaotic	2D mesh	16	20	3	132
RDT	320000	111400	wormhole, 2v	3D RDT	18	16	2	299
RR	600000	625000	adaptive, 5v	2D mesh	75	16	5	300

Table 3: Important cost factors for router chips. In the *Type* column we give the number of virtual channels where necessary, e.g $2v$ means 2 virtual channels. The second and third columns compare the actual number and the estimated number of transistors. With *Flits* we show the flit size or the number of bits transferred in one cycle. *FIFOl* shows the length of FIFOs in flits and *SetofQ* shows the set of queues in the design often reflecting the number of virtual channels.

dred to several hundred pins, we assume that in 10-12 years packaging will allow no more than roughly 2000 pins. The maximum possible global bandwidth is then $b_{max} = 2000/W$. The global communication cost factor, $K_{bs} = 10^5$ multiplied with the wordsize is approximately the cost in SRAM bit equivalents of one word per cycle of global I/O bandwidth.

$$K_{bg}(b_g) = \begin{cases} \infty & \text{if } b_g > b_{max} \\ B_{bg} + K_{bsg} W b_g & \text{otherwise} \end{cases} \tag{7}$$

Global latency cost. For simplicity we assume this cost as constant reflecting the more or less constant speed of DRAM access over time. B_{lg} is estimated at 10^5.

$$K_{lg}(l_g) = B_{lg} \tag{8}$$

Total cost of the system. The total cost of the system is equal to the sum of its components.

$$K(x) = P(K_p(p)+K_c(c)+K_m(m))+K_{bg}(b_g)+K_{lg}(l_g) \tag{9}$$

2.5 Application model

The application model contains functions and parameters that are necessary for application performance characterization. To predict the performance of an application with a particular machine configuration, we assume that the resource demands are uniform over time and that processing, local and global communication can be completely overlapped. Some algorithms, such as those used in dynamic programming, also require the estimation of the algorithmic imbalance or the idle time due to synchronization overhead. Applications with several phases can be handled by dividing the application into its phases and characterizing each phase separately. Our assumption that processing, local and global communication are overlapped imposes constraints on how the problem is partitioned and on the total amount of memory required. As we will show later, besides the memory needed to hold the problem, local and global communication buffers are required in order to be able to overlap communication times.

We will exemplify the concepts of this section by analyzing the Jacobi relaxation problem. The requirements of the other applications considered in this paper are presented in the Appendix. The Jacobi Relaxation problem is an iterative algorithm which, given a set of boundary conditions, finds discretized solutions to differential equations of the form $\nabla^2 A + B = 0$. Each step of the algorithm replaces the value at each node of a grid with the average of the values of its nearest neighbors.

The original Jacobi problem defined by a grid of size N is partitioned in subgrids of size N' as illustrated in Figure 5. Each subgrid or subproblem is solved by storing the subproblem of size N' in the internal memory of a Raw microprocessor and running a blocking relaxation algorithm. After a given number of phases, the subgrid is stored in external RDRAM, and the next subgrid is loaded. Clearly, a given subgrid has to be loaded and operated upon multiple times to reflect the effect of synchronization with the values computed in neighboring subgrids.

Because, values from neighboring subgrids do not impact the relaxations on a given subgrid stored in the microprocessor, the number of iterations needed for convervence increases. We choose $i_s = \sqrt{N'}/2$ as the number of iterations after which resynchronizations must occur between subproblems. Starting with some boundary conditions this means propagating border values to all points in a subproblem. We chose the total number of iterations as being $i_t = N^2$ giving an error reduction factor of ten.

Let us analyze the requirements of this application.

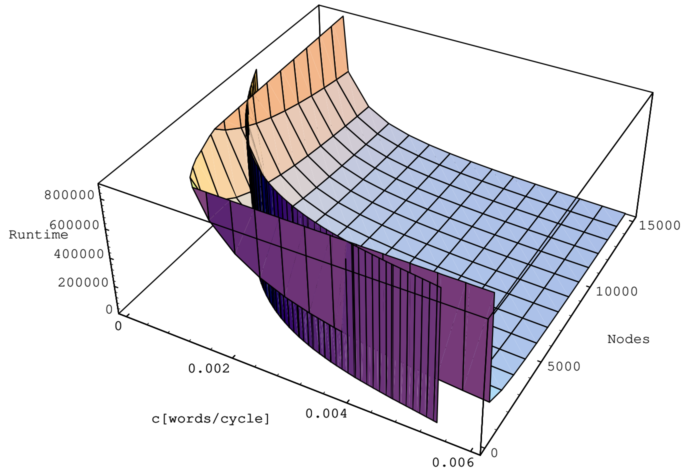

Figure 6: Graphical illustration of the solution to the optimization problem; example shown is for Jacobi Relaxation (one iteration), gridsize 4000×4000 points, software overhead is zero, and global latency equals 100 cycles. The two surfaces correspond to the runtime performance function and a combined equation for the constraint functions. The intersection of the two surfaces determines the points that give balanced machine configurations. The points (c_{opt}, P_{opt}) that correspond to the smallest runtime are the global optimal solutions of the optimization. Other parameters such as processing power and memory size can be determined from the constraint equations by substitution.

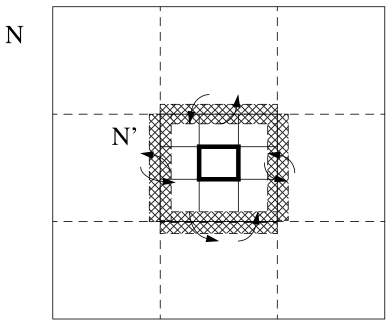

Figure 5: Jacobi Relaxation. The problem of size N is first partitioned in subproblems of size N'. Each subproblem is solved with blocking on P processors. Each processor receives bordering data from its four neighbors and sends its data along borders to its neighbors. Subproblems are resynchronized after a number of iterations.

Required processing per node R_p. This requirement reflects the total amount of computation required per Jacobi node given the algorithmic assumptions described above. The total number of operations for each point are three additions and one multiplication.

$$R_p = i_t 4 \frac{N}{P} = 4 \frac{N^3}{P}. \quad P \in [1, N] \tag{10}$$

Required amount of memory words per node R_m. The required memory is comprised by the memory required to solve the subblock of size N' and also the memory buffers needed to overlap local and global communication.

$$R_m = \frac{N'}{P} + 4\sqrt{\frac{N'}{P}} + 2\frac{N'}{P} = 3\frac{N'}{P} + 4\sqrt{\frac{N'}{P}}. \tag{11}$$

Required number of words of local communication per node R_c. The required local communications is the total amount of data sent or received during the whole execution time. For any iteration each processor requires the bordering points from its neighbor processors.

$$R_c = i_t \times 8\sqrt{\frac{N'}{P}} \times \frac{N}{N'} = 8\frac{N^3}{\sqrt{N'P}} \tag{12}$$

Required local communication events R_o. These events incur a software penalty for initiating a communication step. It reflects the total number of times a local send or receive is issued.

$$R_o = R_c \times \frac{1}{\sqrt{\frac{N'}{P}}} \tag{13}$$

Required latency of events R_l. Reflects the total number of times a local send is issued.

$$R_l = R_c \times \frac{1}{2\sqrt{\frac{N'}{P}}} \tag{14}$$

Required global communication R_{bg}. Reflects the total amount of words of global communication per chip.

$$R_{bg} = \frac{i_t}{i_s} 2N = 2N^2\sqrt{N} \tag{15}$$

Required global communication events R_{lg}. Reflects the total times global sends or reads are initiated per chip.

$$R_{lg} = \frac{R_{bg}}{N'} \tag{16}$$

2.6 Performance Functions

The performance functions estimate the running time of an application in terms of application requirements and architecture parameters.

Runtimes $< T, T_p, T_c, T_g >$. Let the times for processing, local communication, and global communication be T_p, T_c, and T_g respectively. Under the assumptions that local and global communication time are overlapped with computation, the parallel runtime is defined as the maximum of these times.

$$\begin{aligned} T &= max(T_p, T_c, T_g) \\ T_p &= \frac{R_p}{p} + R_o o + R_{lg} o \\ T_c &= \frac{R_c}{c} + R_l k_d l \\ T_g &= \frac{R_{bg}}{b_g} + R_{lg}\frac{k_d}{2} l + R_{lg} l_g \end{aligned} \tag{17}$$

As an example, if the number of operations that must be performed is R_p and the processing power is p operations per cycle, then the processing time is simply R_p/p. Similarly, if the number of events incurring the message overhead (o cycles) is R_o, then the time wasted in message overhead activity is $R_o o$.

2.7 The optimization problem

In this section we describe in more detail the optimization procedure. The optimization procedure is also illustrated graphically in Figure 6.

The problem solved is the following *constrained based nonlinear optimization problem:*

Given: a fixed chip area or budget B and a problem size N

Objective:

$$min\,(T(N, N', x)) \tag{18}$$

subject to the constraints given below, where x is a specific machine configuration $< p, P, m, c, o, l, b_g, l_g >$. The solution of this optimization is the optimal machine configuration: $x_{opt} =< p, P, m, c, o, l, b_g, l_g >_{opt}$ and and the optimal subproblem size: N'_{opt}

Constraints:

1. Budget B must be greater or equal than the total cost. The total cost of the system is computed as the sum of its components.

$$B \geq P(K_p + K_c + K_m) + K_{bg} + K_{lg} \tag{19}$$

It is expedient to use an additional set of balance constraints as given below, when the communication and computation are overlapped. The balance constraints focus the search for the optimal solution to balanced machine configurations. In other words, second and third equations state that communication and computation times should be equal. If they are not equal, we can take resources from the faster component without increasing runtime. The fourth balance constraint states that the memory should fit the problem. If the memory is larger than this amount, it can be reduced without impacting performance. When local, global communication times are equal and memory fits the problem, the machine configuration is balanced for the application. In a balanced machine each resource is utilized to its fullest. The balance constraints greatly reduce the search space, and thus the complexity of the optimization procedure.

2. Balanced local communication with computation.

$$T_p = T_c \tag{20}$$

3. Balanced global communication and computation.

$$T_g = T_p \tag{21}$$

4. Memory on processor element must fit the memory required for a block. Besides the memory required for actual computations R_m^a, buffers for local and global communications R_m^l, R_m^b are allocated because of overlapping conditions.

$$m = R_m = R_m^a + R_m^l + R_m^g \tag{22}$$

3 Analysis

In this section, we study a set of applications in the context of the framework presented. The applications are: Jacobi Relaxation, Dense Matrix Multiply, Nbody, FFT, Largest Common Subsequence. We chose these applications becouse they are diverse and require conflicting machine performances to run efficiently. The optimization procedure has been implemented in Mathematica. We use a 3 cycle software overhead and a 100 cycle DRAM access latency. We also counted an 8Kbyte SRAM-based instruction cache per node.

In all the experiments we used a budget of 1 billion SRAM bit equivalents or the area required for 1 billion logic transistors. This budget is achievable in 10-12 years as projected by the Semiconductor Industry Association (SIA) given a 10-20% growth rate per year of die areas and a growth rate in transistor counts of between 60 and 80% per year due to increasing densities.

Application specific results. Figure 7 shows the optimal division of chip resources for the various applications as a function of problem size. The optimal amount of each resource is shown in greater detail in Figures 8 through 12. Table 4 summarizes the optimal configurations and chip sizes.

Perhaps the most important result from Figure 7 is that the amount of area devoted to processing and local communication is more or less constant at about 75 percent for all the applications and problem sizes.

The global communication bandwidth of 30 words per cycle is the maximum achievable given a packaging technology allowing 2000 pins. The only application that is I/O limited and requires this bandwidth is FFT. All the other applications have a negligible area allocated to global communication. The total chip area for global communication is relatively small even for FFT. Therefore, providing the maximum possible global bandwidth is not a bad idea in a final configuration.

As we can see, the relative communication area required is small in applications such as Jacobi and LCS as they also show good spatial locality. These applications can use most of the resources for processing. FFT and Nbody require the largest communication area with an optimal communication bandwidth between 4 and 5 words per cycle. The division between processing and memory areas is uniform.

The matrix multiplication based on Connor's memory efficient blocking algorithm gives the most uniformly divided configuration. For this application, memory, local communication and processing areas are approximately equal.

The amount of memory per node obtained is relatively small compared to modern day multiprocessors in all applications. The reason is twofold. First, the total amount of memory in the entire Raw chip is still quite large, since it is the product of P and m. Second, fast local communication obviates the need for huge amounts of local memory. The matrix multiplication required the largest amount of memory giving a total of 24 Kbytes per node. The smallest memory is required for Nbody.

For all the applications the optimal processing power obtained is equivalent to a single-issue processor. The total number of processors P varied between 1100 to 2310 for large problem sizes.

Although the optimal machine configurations obtained vary for different applications, problem sizes and budgets, the general trends for various applications are similar. Accordingly, for the applications studied, assuming an 1 billion logic transistor equivalent area, we recommend building a Raw chip with approximately 1000 tiles, 30 words/cycle global I/O, 20Kbytes of local memory per node, 3-4 words/cycle local communication bandwidt, and single-issue processors. This configuration will give performance near the global optimum for most applications.

Sensitivity of grain size. The framework helps answer many other questions about machine configurations. Let us study the *sensitivity* of performance to

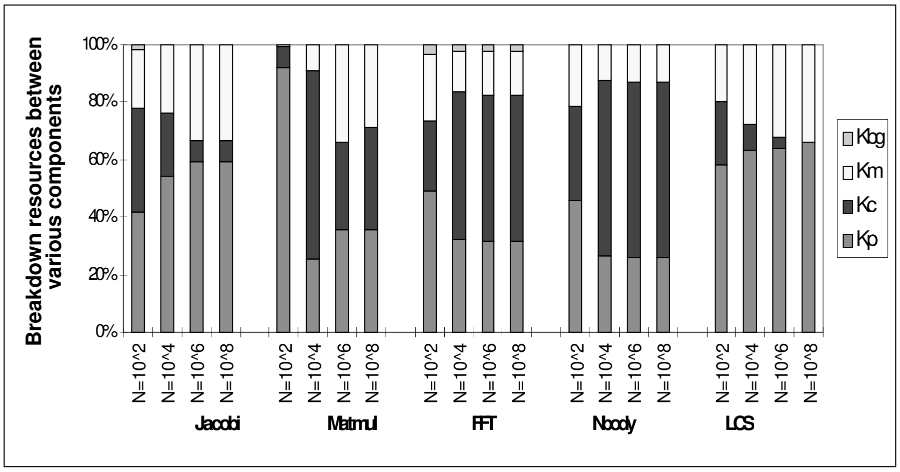

Figure 7: Breakdown of chip areas for processing, memory, local communication and global communication that give optimal machine configurations for a budget of 1 billion logic transistor equivalent area.

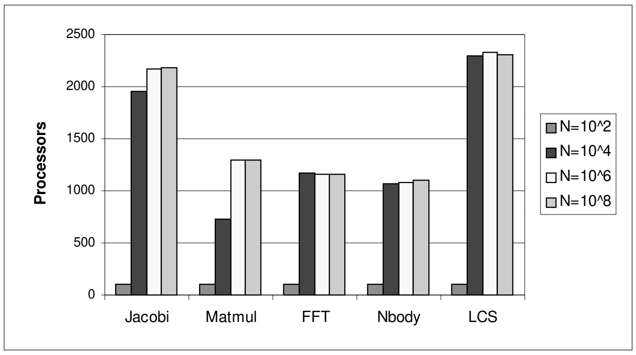

Figure 8: Number of processors in optimal machine configurations for different problem sizes.

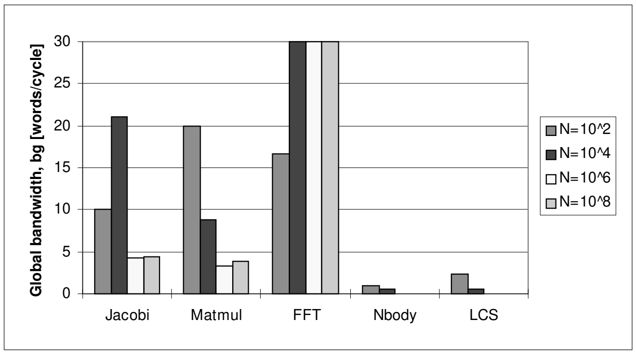

Figure 9: Global io bandwidth b_g in optimal machine configurations for different problem sizes.

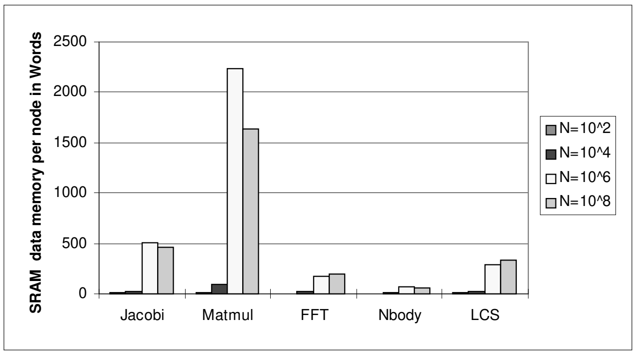

Figure 10: Local SRAM data memory m per node in optimal machine configurations.

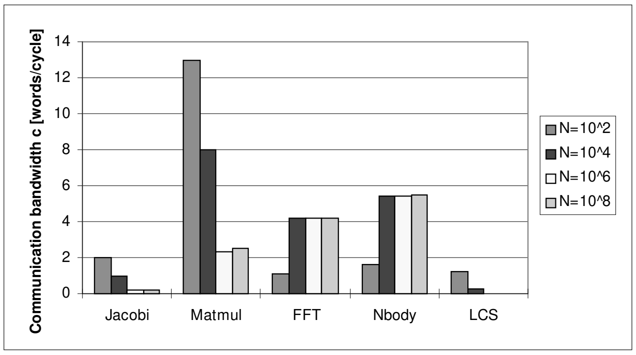

Figure 11: Local communication bandwidth c in optimal machine configurations for different problem sizes.

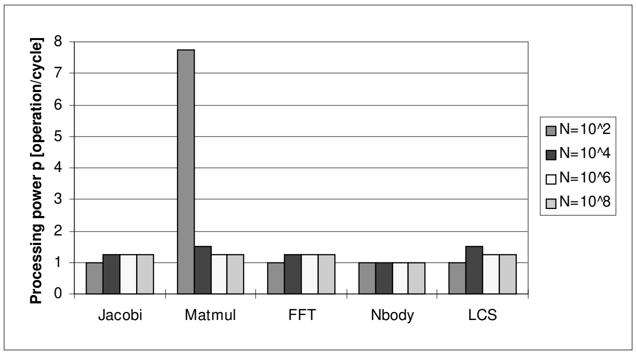

Figure 12: Processing power p in optimal machine configurations for different problem sizes.

the machine configuration near the optimum machine configuration point. This study is useful to determine a machine configuration that is robust across many applications. As an example, let us determine the machine configuration with the smallest number of nodes whose performance is within 25 percent of the optimal configuration.

Results are shown in Table 5. For each application, the first row gives the optimal configuration. The second row gives the configuration with the smallest number of nodes under the condition that the performance is no worse than 25 percent of the optimal. As we can see, balanced machine configurations with less nodes usually take advantage of the parallelism available in superscalar processors. However, for all the applications studied the configuration that gave best performance used nodes based on 2-way superscalars at most.

Design comparisons. The framework also allows us to compare competing designs for the same budget. As an example, let us compare the two designs: (1) using on-chip SRAM and routers with 16flit FIFOs, and (2) using only a small SRAM cache and the rest of memory in on-chip DRAM as well as small 2flit FIFOs. We derive the performance/cost optimal configurations and look to application performance for different problem sizes.

Since DRAM densities are much higher than SRAM densities we can have more memory per node in alternative (2). One problem in using DRAMs is that the access latency that is much higher than corresponding SRAMs. To reduce the impact of the latency, we include a small SRAM cache in each node and assume that the SRAM cache results in a near perfect hit rate. Case (2) also has small FIFOs – With good static scheduling of the communication channels the need for deep FIFO's is reduced.

The question is how much do these changes impact the performance of applications given performance/cost optimal partitioning of resources in both cases? Figure 13 shows the performance ratio between the second and the first designs. It is easy to see that the larger amount of on-chip memory in case (2) results in significantly higher performance.

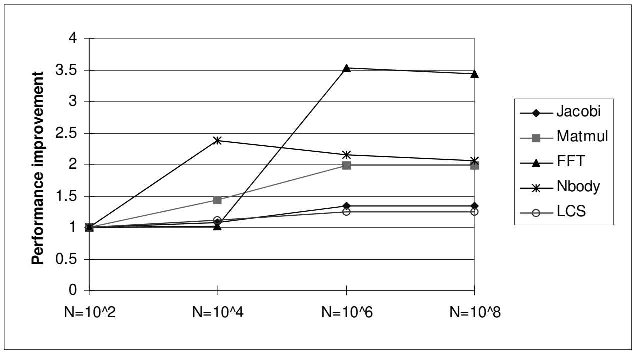

Figure 13: Performance comparison between two cost optimal designs each with a budget of 1 billion logic transistor area. In the first design we use SRAM and routers with 16 flits FIFOs, while in the second design we use on-chip DRAM with a 1Kbyte SRAM cache per node and 2 flit FIFOs.

4 Conclusions

This paper provides a framework for reasoning about single chip microprocessors such as Raw with replicated, fine-grain processing elements. The framework uses a machine characterization that considers processing, memory, local and global communication, and latency as separate machine resources. This is a unique characterization of machine space since it captures the effects of locality by treating local and global communication separately. The framework incorporates a cost model based on empirical observations and statistics gathered on current implementations of superscalars and router chips.

The framework recognizes the importance of balance in good design, and integrates this idea with a cost and performance model to provide a useful design tool. Having provided this framework, this paper chooses a diverse application suite in order to exercise the framework and to address some general questions

Problem	Size	P	c	p	m	b_g		PK_p	PK_c	PK_m	K_{bg}	K_{lg}
Matmul	10^8	1290	2.6	1.25	1640	3.9		35	35	28	0.02	0.01
Matmul	10^6	1290	2.3	1.25	2230	3.3		35	30	33	0.02	0.01
Matmul	10^4	724	8	1.5	97	8.8		25	65	9	0.05	0.01
Jacobi	10^8	2180	0.19	1.25	464	4.4		60	7.4	32.6	0.03	0.01
Jacobi	10^6	2171	0.19	1.25	502	4.2		60	7.4	32.6	0.03	0.01
Jacobi	10^4	1950	1	1.25	25	21		53	22	23	0.16	0.01
Nbody	10^8	1100	5	1	61	0.06		26	60	13	0.0004	0.01
Nbody	10^6	1080	5	1	67	0.06		26	60	13	0.0004	0.01
Nbody	10^4	1070	5	1	8	0.5		26	60	13	0.004	0.01
FFT	10^8	1160	4.2	1.25	178	30		31	50	15	0.2	0.01
FFT	10^6	1160	4.2	1.25	178	30		31	50	15	0.2	0.01
FFT	10^4	1160	4.2	1.25	178	30		31	50	15	0.2	0.01
LCS	10^8	2310	0.01	1.25	337	0.015		63	3.7	32	0.0001	0.01
LCS	10^6	2330	0.01	1.25	291	0.014		63	3.7	32	0.0001	0.01
LCS	10^4	2290	0.25	1.5	20	0.25		53	9	27	0.002	0.01

Table 4: Breakdown of resources and optimal machine configurations for three problemsizes. Columns P to b_g represent the optimal machine configuration and the columns from PK_p to K_{lg} are the chipsizes in percent of the total cost.

Problem	P	N'	c	p	m	b_g	PK_p	PK_c	PK_m	K_{bg}	K_{lg}
Matmul	1290	548x548	2.6	1.25	1640	3.9	35	35	28	0.01	0.01
(18%)	826	685x685	1.7	2	3975	2.5	46	20	32	0.01	0.01
Jacobi	2180	302230	0.19	1.25	464	4.4	60	7.4	32.6	0.01	0.01
(24%)	811	302230	0.3	2	538	5	77	5.5	16	0.01	0.01
Nbody	1100	8306	5	1	61	0.06	26	60	13	0.0002	0.01
(21%)	799	295145	5	1.5	2954	0.001	28	47	24	0.00001	0.01
FFT	1160	68718	4.2	1.25	178	30	31	50	15	0.2	0.01
(23%)	356	3518440	2.7	1.75	29594	30	17	10	71	0.2	0.01
LCS	2310	193427	0.01	1.25	337	0.015	63	3.7	32	0.0001	0.01
(23%)	1119	4354	0.05	1.75	499	0.015	73	2.4	23	0.0001	0.01

Table 5: Solutions that come within 25% of the optimal for a problem size of 10^8 with the smallest number of nodes P. The first row of each application shows the global optimum, and the second row shows the solution with the minimum number of processors and performance within 25% of the optimal. The numbers in parentheses show the performance degradation compared to the global optimum for the configurations with the minimum processors. The first columns between P to b_g represent the optimal machine configuration and the columns from PK_p to K_{lg} are the chipsizes in percent of the total cost.

in parallel computer design in general. More specifically, it addresses the questions of on-chip resource division in the MIT Raw microprocessor.

Although the optimal machine configurations vary for different applications, problem sizes and budgets, the general trends are consistent. The framework recommended that chip designers devote about 75 percent of the chip area to processing and local communication. The framework further suggested that for the applications studied and assuming an 1 billion logic transistor equivalent area, designers should build a system with about 1000 nodes, 30 words/cycle of global I/O, 20Kbyte of local memory per node, 3-4 words/cycle local communication bandwidth and single-issue processors for optimal performance.

5 Acknowledgements

This work leverages the early work on grain size by Yeung, Dally, and Agarwal [5]. The research is funded by DARPA contract #DABT63-96-C-0036. We are also grateful to Tom Knight for many relevant discussions on cost modeling.

A Applications

References

[1] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal Baring it all to Software: Raw Machines. *IEEE Computer*, September 1997, pp. 86-93.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Proc. of Fourth ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming,* May 1993.

[3] A. Alexandrov, M. Ionescu, K. E. Schauser, and C. Scheiman. "LogGP: Incorporating Long Messages into the LogP Model" *Proc. of the SPAA'95*, Santa Barbara, CA, July 1995.

[4] J. Babb and R. Tessier and M. Dahl and S. Hanono and D. Hoki and A. Agarwal. Logic Emulation with Virtual Wires. *IEEE Transactions on Computer Aided Design*, VOL. 16, No.6, June 1997, pp. 609-626.

[5] Donald Yeung, William J. Dally, Anant Agarwal. How to Choose the Grain Size of a Parallel Computer. *MIT/LCS Technical Report, MIT-LCS-TR-739.*

[6] Charles L. Seitz, Nanette J. Boden, Jakov Seizovic, and Wen-King Su. The Design of the Caltech Mosaic C Multicomputer. *Research on Integrated Systems, Proceedings of the 1993 Symposium*, The MIT Press. Cambridge, Massachusetts, 1993. pp. 1-22.

[7] Hiroaki Nishi, Ken-ichiro Anjo, Tomohiro Kudoh, Hideharu Amano. The RDT Router Chip: A Versatile Router for Supporting Shared Memory. *Special Issue on Architecture, Algorithms and Networks for Massively Parallel Computing*, IEICE, VOL. E00-A, No.1 January 1997.

[8] CPU Info Center. *http://infopad.eecs.berkeley.edu/CIC/tech/*

[9] Peter R. Nuth and William J. Dally. The J-Machine Network, *Proceedings of the 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 1992. pp. 420-423.

[10] Charles L. Seitz and Wen-King Su. A Family of Routing and Communication Chips Based on the Mosaic. *Research on Integrated Systems, Proceedings of the 1993 Symposium*, The MIT Press. Cambridge, Massachusetts, 1993. pp 320-337.

[11] H. T. Kung. Memory Requirements for Balanced Computer Architectures, *IEEE* 1986. pp. 49-54.

[12] Thomas J. Holman and Lawrence Snyder. Architectural Tradeoffs in Parallel Computer Design. *Advanced Research in VLSI, Proceedings of the 1989 Decennial Caltech Conference*, The MIT Press. Cambridge, Massachusetts, March 1989. pp. 317-334.

[13] Paul Chow The MIPS-X RISC Microprocessor, *Kluwer Academic Publishers*, August 1989.

[14] William J. Dally Architecture of a Message-Driven Processor, *Proceedings of the 14th Annual Symposium on Computer Architecture*, June 1987, pp. 189-196.

Application	R_p	R_c	R_l	R_o	R_m	R_{bg}	R_{lg}
Jacobi	$4\frac{N^3}{P}$	$8\frac{N^3}{\sqrt{N'P}}$	$4\frac{N^3}{N'}$	$8\frac{N^3}{N'}$	$3\frac{N'}{P} + 4\sqrt{\frac{N'}{P}}$	$2N^2\sqrt{N}$	$2N\frac{^2\sqrt{N}}{N'}$
Matmul	$2\frac{N^3}{P}$	$4\frac{N^3}{N'\sqrt{P}}$	$2\frac{N^3\sqrt{P}}{N'^3}$	$4\frac{N^3\sqrt{P}}{N'^3}$	$7\frac{N'2}{P}$	$2\frac{N^3}{N'} + N^2$	$2\frac{N^3}{N'^3}$
Nbody	$2\frac{N^2}{P}$	$2\frac{N^2}{P}$	$\frac{N^2}{P}$	$2\frac{N^2}{P}$	$\frac{N'}{P}$	$4\frac{N^2}{N'}$	$\frac{N^2}{N'^2}$
FFT	$12\frac{N}{P}\log N$	$2\frac{N}{P}\log N$	$\frac{N}{N'}\log N$	$2\frac{N}{N'}\log N$	$4\frac{N'}{P}$	$2N\log N$	$2\frac{N}{N'}\log N$
LCS	$2\frac{N^2}{P}$	$2\frac{N^2}{N'}$	$\frac{N^2}{N'}$	$2\frac{N^2}{N'}$	$4\frac{N'}{P}$	$4N$	$\frac{N}{N'}$

Table 6: Overview application requirements.

[15] William J. Dally and Charles L. Seitz. The Torus Routing Chip, *Distributed Computing*, Volume 1. pp. 187-196.

[16] Anant Agarwal, David Chaiken, Godfrey D'Souza, Kirk Johnson, David Kranz, John Kubiatowicz, Kiyoshi Kurihara, Beng-Hong Lim, Gino Maa, Dan Nussbaum, Mike Parkin, and Donald Yeung. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. *Proceedings of the Workshop on Scalable Shared Memory Multiprocessors.* Kluwer Academic Publishers, 1991. Also appears as MIT/LCS Memo TM-454, 1991.

[17] William J. Dally et al. The J-Machine: A Fine-Grain Concurrent Computer, Proceedings of the IFIP (International Federation for Information Processing), 11th World Congress, *Elsevier Science Publishing*, New York, 1989. pp. 1147-1153.

[18] CM5 Technical Summary, Thinking Machines Corporation, Cambridge, MA. Oct, 1991.

[19] CRAY T3D System Architecture Overview, Cray Research, Inc. Revision 1.C, September 23, 1993.

[20] Keith Diefendorff and Michael Allen. Organization of the Motorola 88110 Superscalar RISC Microprocessor, *IEEE Micro*, Volume 2, Number 2, April 1992. pp. 40-63.

[21] Dennis Allison and Michael Slater. National Unveils Superscalar RISC Processor, *Microprocessor Report*, Volume 5, Number 3, February 20, 1991.

[22] Daniel Dobberpuhl et. al. A 200 Mhz 64b Dual-Issue Microprocessor, *IEEE Solid State Circuits Conference*, Volume 35, February 1992. pp. 106-107.