Automatic Exploration and Characterization of the Cluster Design Space

W. R. Dieter and H. G. Dietz

dieter@engr.uky.edu hankd@engr.uky.edu

Electrical & Computer Engineering Department

University of Kentucky

Lexington, KY 40506-0046

Abstract

A cluster is a parallel supercomputer built using interchangeable parts. Not only do many of the components benefit from commodity pricing, reducing system cost, but it also becomes practical to customize the architectural configuration with a breadth of choices that was unimaginable just a few years ago. At SC2002, we introduced the Cluster Design Rules (CDR), a software tool which helps users design clusters by automatically generating all designs their parts database can support and evaluating them, in detail, based on user-supplied performance parameters ranging from computational properties like network latency to physical parameters like cost and power consumption. As the WWW-based CDR has been widely used to create thousands of optimized cluster designs, we have made it freely available as full source code, and have continued to improve it.

This paper describes the concepts and implementation methods used in the CDR, also presenting an entirely new type of use: the ability to determine high-level properties of the cluster design space by evaluating how the properties of the set of designs meeting all constraints changes as the constraints are changed. Examples include: how network designs are influenced by changes in system cost, latency, and bisection bandwidth requirements; the tradeoffs between uniprocessor and multiprocessor nodes; and how cooling capacity affects design options.

While it is fairly easy to put together a cluster supercomputer using unlimited resources, most cluster buyers have limited money, space, and electric power. Commodity supercomputer clusters offer an outstanding price/performance ratio, but getting the best performance with a limited budget is a difficult problem. After ten years of building dozens of PC clusters, we have evolved a relatively complete set of basic engineering principles for design, construction, and use of "Beowulf" supercomputers. Our first attempts to codify these cluster design rules were in the traditional academic forms: research publications and more accessible forms of technical papers (e.g., the Linux Documentation Project's *Parallel Processing HOWTO[6]* and an article for Ars Technica[8]). However, the rules are too complex to be assembled into a single easily understood document. We needed a way to let people create expert-quality designs without becoming experts.

The answer is to build an expert system software tool. However, the problem of optimizing system design is largely a matter of achieving the best balance between engineering tradeoffs, and traditional expert systems are not well suited to expressing formula-driven optimization problems. For example, the best choice of processor is not always the fastest processor, nor the one with the best peak performance per unit cost, nor the cheapest; the best processor is the one that is used in the design that best meets the requirements determined by your application(s) and budget (e.g., money, space, power consumption). Thus, we built a somewhat unusual type of expert system that performs a pruned exhaustive search of the cluster design space, evaluating each potential design according to constraints given by or derived from the user. The expertise is primarily in the form of the tool searching a much larger design space than a novice would have realized exists, but also is evident in the formulas used to evaluate total system performance.

Many cluster designers have addressed either the price or the performance half of the price/ performance equation, but not both simultaneously. The Cluster Design Rules (CDR) tool evaluates performance based on traditional high performance computer design, (see on page 10) while considering price from a database based on Internet pricing services like `PriceWatch.com`, `Froogle.Google.com`, and `PriceGrabber.com`.

The CDR fulfills several needs. It helps inexperienced designers decide what to buy for their clusters. An experienced cluster designer can explore the design space more quickly and thoroughly using the CDR than by doing

so by hand. Such design space explorations can yield interesting insights that are not immediately obvious, even to experienced cluster designers. Even so, judgment is still required to capture the proper requirements and apply the results.

The CDR is accessed via a WWW form linked from `http://aggregate.org/CDR`, although anyone can make their own WWW form to serve as the user interface because the source code is distributed through the BDR project at SourceForge.net. As a matter of policy, we do not record details about people's use of the tool. However, through personal contacts, we know that the tool has been used to help design a large number of clusters.

1 The CDR User Interface

The goal of the CDR is to efficiently search all possible system configurations to identify which designs are most appropriate for a particular application or class of applications. The web form asks a series of high-level questions to determine minimum system requirements and user-specified criteria by which designs meeting those requirements will be ranked.

Automating the design process requires balancing competing requirements. The tool must be given sufficient detail to generate results that embody significant engineering insight, yet the complexity and bulk of the input requested from the user must be minimal. Our desire to implement the tool using an interactive web interface imposes additional constraints. We were pleasantly surprised to discover that, using relatively simple input, our tool was able to create good designs that most human designers would not have considered. Most designers consider a small range of configurations relative to questions like, "How many FLOPS does my application need?" or "How much memory does my application need?" The tool considers a larger search space and also asks a more complete set of questions. While the input is not detailed enough to produce performance estimates accurate within a few percent of actual performance, it does give results that are "in the ball park". Such simple models have been successfully used before[21].

The web form relates performance characteristics to a unit of computational work. For convenience, the form refers to these units as FLOPS (floating-point operations per second) because the computational work in many scientific applications is dominated by floating point calculations. The form requests application information about memory, disk, network, and processor usage, as well as resource constraints of the site where the cluster will be housed. Application information can be obtained by analyzing the code by hand or using automated tool[3].

1.1 Memory

Memory size and speed are important criteria for many parallel applications. The CDR form begins by asking how much memory the application needs per node for the operating system and application code, and how much the entire cluster needs for application data.

After memory size requirements, the CDR asks for the memory bandwidth requirement. Main memory bandwidth limits performance more than floating point operations for parallel applications that do not fit in cache. The CDR measures main memory bandwidth in GB/s per peak GFLOPS, or bytes/FLOP, rather than GB/s because the number of bytes accessed per FLOP is an intrinsic property application program. Sometimes the CDR chooses cluster designs with "slow" processors for memory bandwidth limited applications because the slow processors are fast enough to keep up with the memory subsystem and less expensive. In these cases, money saved by using slow processors can be spent on more processors, each with its own path to memory. Overall, the aggregate memory bandwidth of the system is increased along with the total number of FLOPS.

1.2 Disks

Nodes may need disk drives for swap space or local data storage. In workstations swap space is often used to extend main memory. Relying on swap space to provide enough memory for the application generally leads to poor performance; even the slowest main memory is orders of magnitudes faster than disk. Swap space can be useful, however, when running software packages that "leak" memory: software that fails to deallocate memory after it is no longer in use. Ideally, the leaky software should be fixed, but that is not always feasible.

Out-of-core applications explicitly manage data movement between main memory and a large data set stored on a local disk. For such applications, the CDR can include disks for each node in the cluster design. If data stored on local disks must be able to survive a disk failure, redundant local disks can be required.

1.3 Network

Configuring the network is one of the most complex, but important parts of designing a cluster supercomputer. Application requirements for the network are described in terms of latency and bisection bandwidth per processor. Latency is the amount of time required for one processor to receive a message from another processor. Applications in which processes synchronize frequently often need low latency. The CDR asks for latency both in terms of process-to-process latency and in terms of latency for collective communication patterns, like barrier synchronization and reduction operations.

Bisection bandwidth per processor measures the amount of data, per processor, between two halves of the network when all processors are sending messages at the same time with the network is cut in half such that the bandwidth between the halves is minimized. Bisection bandwidth is a better measure of cluster interconnect bandwidth than aggregate bandwidth, the sum of the bandwidth of all the links in the network, because it accounts for congestion from multiple nodes sending messages at the same time. The CDR measures bisection bandwidth per processor, rather than per node, so that each processor will be able to get its full share of bandwidth.

1.4 Physical Requirements

Physical requirements, such as number of nodes, size, power consumption, and cooling are just as important as requirements like FLOPS, network bandwidth, and network latency. Power and cooling are becoming increasingly important as the amount of power consumed by a single CPU increases. If the cluster room is too small or does not have enough power or cooling the cluster is unusable. One commonly forgotten parameter is the number of spares. Spares are a requirement, no matter how reliable the vendors or how good the maintenance contract. Even if a vendor is willing to ship products overnight, identical parts may not be available six months after the cluster is built. The CDR can evaluate designs using either hot spares, that are connected and running all the time, or cold spares, which are powered down and substituted for nodes as they fail.

The total number of nodes should not be a design parameter. It should instead be derived from the other requirements of the system. The CDR does allow the user to specify mathematical constraints on the number of nodes. For example, some problems only work well with a perfect square, a perfect cube, or a power of two number of nodes (or processors).

Users are often surprised by how large a cluster they can really afford. Size, power, and cooling constraints can quickly limit the size of viable cluster supercomputers, so the CDR keeps track of the size of designs in 2' by 2' squares, how much current they draw in amps, and how many tons of air conditioning are required to remove the heat generated by the cluster from the rooms.

1.5 Floating Point Performance

The final requirements the CDR asks are budget and number of GFLOPS the cluster must achieve. Though GFLOPS is the first thing people think of, and they are important, the CDR asks GFLOPS last to reinforce the idea that there are many other important parameters.

1.6 Metrics

The CDR helps designers sort through designs that meet their requirements by allowing the user to specify weightings for the criteria of the design. The values for each criteria in a viable design are computed as a percentage of the specified value and multiplied by the user-specified weightings to compute a score for the design. For example, assume the user gives memory size a weighting of 10 and GFLOPS a weighting of 1. If the cluster is required to have at least 1 TB of memory and 500 GFLOPS of peak performance, then a design that has 1.5 TB of memory and 600 GFLOPS would have a metric value of $\frac{1.5TB-1.0TB}{1.0TB} \times 10 + \frac{600GFLOPS-500GFLOPS}{500GFLOPS} \times 1 = 5.2$. Designs that meet the base requirements for a design are sorted by their metric values, and those with the highest values are displayed. Designs with equal metric values are displayed in an arbitrary order.s

1.7 Component Limitations

Sometimes organizations have exclusive contracts with suppliers or are able to take advantage of bulk pricing. To support exclusive contracts, the CDR lets the user either modify the parts database, or more simply, enter names of

parts that should be included or excluded from every design.

2 The CDR Tool Logic

While the CDR web form provides helps designers to specify their requirements, the logic embodied in the Common Gateway Interface (CGI) program searches for the best designs matching those requirements. The CDR generates designs by trying every possible combination of hardware available in its components database using a pruned depth first search. Components are added to the design one at a time in a predefined order, each component representing a branch in the search tree. The CDR evaluates the current design as each component is added.

2.1 Pruning the Search Space

Searching all possible combinations of components would be computationally infeasible without aggressive pruning. The CDR prunes the search space based on system requirements, cost, and a user supplied metric. Requirements pruning discards designs that do not meet the requirements a user entered on the CDR search form. For example, if a design does not have enough processing power to meet the GFLOPS requirement, it can be pruned without checking network designs or any other system components.

As a special case of requirements pruning, the CDR also prunes designs based on cost. As soon as the cost of a partial design exceeds the budget requirement, no combination of parts can be added to create a viable design. In practice, cost pruning often eliminates a large number of design choices early in the search.

Designs that survive requirements and cost pruning are also subject to metric pruning. Though millions of designs may meet a user's requirements, only the top designs are recorded. After a component is added to the design, the CDR computes a partial metric for the selected parts and an upper bound on the metric for the remaining parts. When the partial metric plus the upper bound metric is less than metric of the worst design stored so far, no design based on these parts can be better than the worst recorded design, so the current design is pruned. Like cost pruning, metric pruning greatly reduces the number of designs fully evaluated.

2.2 Searching for Solutions

The design space search starts by finding upper and lower bounds on the number of nodes. The upper bound is the maximum number of minimal cost nodes the budget will allow. The lower bound is the minimum number of nodes that meet both the floating-point performance and memory requirements. The CDR evaluates designs with numbers of nodes between the lower and upper bounds in an order that spreads the search over a wide area of the search space quickly. This strategy typically finds solutions with high metric values early, allowing metric pruning to eliminate a large number of designs that meet the user's requirements, but have poor metric values.

The CDR computes the number of spares from the number of nodes and determines if the current design matches the constraints on number of nodes (e.g., a square, a cube, or some exact number.) Designs that pass the node number constraints are subject to metric pruning.

After determining the number of nodes, the CDR tries each of the available motherboards in turn, pruning based on cost and metric evaluation. The choice of motherboard impacts the choice of almost every other part. By choosing the motherboard early in the selection process the CDR can quickly prune designs that will not work with a particular motherboard because they require too many memory slots, the wrong kind of processor, the wrong kind of memory, etc.

Next, the CDR evaluates network topologies with the current design. The network topology portion of the search space is the most time consuming part to explore by hand. The CDR evaluates designs with no network, direct connections between nodes, a ring, 2-dimensional and 3-dimensional meshes, a single switch, a tree of switches, a flat neighborhood network (FNN), and a FNN of trees. Each of the network topologies have different properties that affect the cost, latency, and bisection bandwidth of the design. For example, the simplest network topology, no network, has low cost and low latency, but only works with one node. Direct connections between nodes has low latency and high bandwidth because no switches are needed, but scalability is limited by the number of PCI slots available.

Other network topologies allow more scalability with varying cost and performance characteristics. The ring (essentially a 1-dimensional mesh), 2-dimensional meshes, and 3-dimensional meshes scale well with cost because the number of neighbors remains constant regardless of the cluster size. However, the latency between any two nodes

increases, as the cluster size increases, and bisection bandwidth increases more slowly than the number of nodes. Connecting all the nodes to a single switch is simple, and effective as long as the cluster is small enough to connect to one switch. A tree of switches allows the cluster to scale to larger sizes, at the cost of latency and bisection bandwidth. FNN's [7, 10, 16] allow cluster size to scale larger than a single switch width, while preserving single switch latency and increasing bisection bandwidth. An FNN of trees can scale to sizes larger than an FNN, while preserving some of the low latency of an FNN.

Once the network topology has been selected, the CDR determines how many times each network connection can be replicated. Thus it tests for channel bonding with as many network interface cards as can fit in a node. Though most people only think of channel bonding with a single switch network, the CDR evaluates channel bonded designs will all the network types it tries. After selecting the number of network interfaces, the CDR prunes based on cost and metric.

The relationship between components selected after the network and the metrics they affect is straightforward. After the selecting a network, the CDR chooses network cables, processor type, memory parts, cases, racks, and disks respectively. The memory bandwidth requirement is used to scale the processor's GFLOPS rating based on the memory parts used. Pruning takes place based on requirements, cost, and the current metric at each step.

3 A Survey of the Cluster Design Space, April 2005

Although the CDR was originally designed only to suggest solutions to a specific design problem, the tool also can be used to systematically evaluate general properties of solutions to a wide range of design problems.

If there are no performance constraints on the network, what kind of network should be used to obtain the highest GFLOPS for the complete system? Clearly, spending more money on the network means having less money to spend on the processors that deliver the GFLOPS. Thus, the obvious answer would be to build a ring, but each node in a ring must have two network interfaces. Depending on the relative cost of switches, it might be cheaper to use a single network interface per node with a switch fabric. Consequently, even this very simple design problem involves a surprisingly complex set of engineering tradeoffs, all highly sensitive to pricing.

Rather than using the CDR to create a single optimized design, we can use the CDR to produce an analysis of how well various alternative designs meet a specific set of design requirements. Each CDR run typically evaluates tens to hundreds of millions of alternatives. Literally, the CDR evaluates all possible designs given the design constraints, system configuration ability of the CDR, and parts database. The CDR can be set to display details about the N best designs rather than just the single best. By running the CDR multiple times, sampling a range of design constraints, the complex relationship between design features and appropriateness for specific design requirements becomes clear.

Figure 1 summarizes the results obtained by using the CDR to study the 1,024 best design alternatives for maximizing GFLOPS with no network performance constraints. Configurations ranging from $2,000 to $2,048,000 in cost were considered. The intuitive answer – to always build a ring – is actually within 10% of the best design's GFLOPS performance throughout the entire design space. However, a ring (constructed using Fast Ethernet technology) is optimal for less than half of the design space evaluated. At the $2,000 price point, a single cheap switch using Fast Ethernet yields a cheaper network than a ring. By $4,000, a ring is the cheapest, but most network topologies are viable. From $8,000 to $256,000, the range of viable topologies narrows and a two-level tree switch fabric constructed using Fast Ethernet yields the system with the highest GFLOPS performance at each price point. From $512,000 to $2,048,000, as switch fabric cost begins to mount, Fast Ethernet ring topology becomes the clearly best design.

There are designs that the CDR did not consider in this analysis; for example, the CDR currently does not know how to construct a tree with more than two levels and fat trees are not considered because they are not legal topologies for commodity switch technologies and are typically outperformed by FNNs using comparable hardware. There also are designs not considered because the parts database used does not contain all possible components but merely a large representative sample. Costs also vary somewhat because special discounts are sometimes given. However, billions of complete system configurations representing the vast majority of currently possible cluster designs were evaluated in detail, so the results can be considered a reasonably accurate "ground truth" summary of how this design space currently is populated. The following subsections similarly summarize a few more controversial aspects of the design space.

3.1 How is Network Design Affected by Latency Constraints?

Communication latency is a very important cluster design parameter. Given networking components with specific latencies, it is impossible to create a network that has lower latency than the sum of latencies of its components along a communication path. Thus, intuitively, one would expect that tight latency requirements would best be met using networking components that have low latencies – i.e., networking hardware designed specifically for use in cluster system area networks (SANs): Myrinet, Quadrics, etc. However, the CDR reveals a much more complex design space.

As a typical reference point, consider clusters constructed to provide point-to-point messaging latency of no more than $50\mu s$ and at least 200Mb/s bisection bandwidth per processor. The $50\mu s$ latency limit significantly constrains network topology using commodity components, since passing through a single switch consumes more than half that time. Further, although 200Mb/s might not sound like it does much more than eliminate 100Mb/s Fast Ethernet, in fact it imposes a very complex constraint that can be difficult to meet even using SAN hardware capable of multiple Gb/s link speeds: bisection bandwidth is not determined by link speed alone, but is very sensitive to topology.

Figure 2 shows the rather surprising way in which the $50\mu s$ and 200Mb/s requirements are satisfied to maximize total system GFLOPS. Despite a slow link speed, Fast Ethernet actually yields the best designs from $2,000 to $128,000 system cost! Up to $16,000, Fast Ethernet is best because both network interfaces and relatively wide switches are cheap enough that channel bonding can be used to multiply the bandwidth to meet the constraint; from $32,000 to $128,000, the system width exceeds that of a single cheap switch, but is met using an FNN constructed from Fast Ethernet components. In that price range, Gigabit Ethernet and SAN hardware easily exceed the network performance constraints, but at too high a cost... leaving less money for nodes and yielding significantly lower system performance. From $256,000 to $1,024,000, Gigabit Ethernet becomes the winning technology, primarily because very wide switches are available at relatively modest cost. However, by $2,048,000, the node counts are too high for even those wide Gigabit Ethernet switches to cover with single-switch latency; SAN technology's ability to meet the latency constraint while passing through multiple switches makes it the only viable choice. Interestingly, however, the SANs are not fast enough to meet the 200Mb/s constraint using an ordinary tree with so many nodes; instead, they must use an FNN of trees (or, with comparable performance, a fat tree).

As the latency constraint is relaxed, the dominance of commodity technologies becomes even more pronounced. Figure 3 shows that, if the latency constraint is removed entirely, the ability to use multi-level switch fabrics extends the commodity technologies at least to the $2,048,000 price point. Fast Ethernet yields the best designs up to $256,000, but beyond that point the FNN would need too many interfaces to maintain the required bisection bandwidth using Fast Ethernet, so an FNN using Gigabit Ethernet becomes the best alternative. Perhaps more surprising is the fact that, as shown in Figure 4, the solution space takes essentially the same form as soon as the latency limit is high enough to allow two-level switch fabrics using commodity technologies. A latency requirement of $100\mu s$ is sufficient to effect this change in the design space. Thus, allowing latency to go from $50\mu s$ to $100\mu s$ yields significantly higher GFLOPS for the same price, although further loosening of the latency constraint has surprisingly little impact. For example, mesh-based designs are conspicuous by their absence from Figure 3.

What happens if we select a latency requirement that a commodity switch cannot meet? Figure 5 shows that the design space changes dramatically as the latency requirement changes from $50\mu s$ to $10\mu s$. Fast Ethernet and Gigabit Ethernet fare well until there are not enough slots to support a fully connected (direct) network, but that number of nodes is reached before $8,000 total system cost. From $16,000 on, only the relatively expensive SAN hardware can offer system performance that continues to scale. Interestingly, the exact same sub-$8,000 winning designs are still viable contenders against the best $16,000 SAN-based designs (as indicated by the yellow block), but there are literally no larger commodity-based network designs to compete with the larger SAN-based designs.

3.2 How is Network Design Affected by Bandwidth Constraints?

Just as latency constraints yielded design spaces significantly different from the predictions of popular wisdom, bandwidth constraints reveal a few surprises. Holding a constant latency requirement of $50\mu s$, consider what happens as bandwidth constraints range from 100Mb/s to 1Gb/s.

With a requirement of 1Gb/s bisection bandwidth per processor, one would expect the best low-end systems to be based on Gigabit Ethernet and high-end ones to be using SAN technology. As shown in Figure 6, that intuition is essentially correct. The winning Gigabit Ethernet designs follow an interesting topology pattern. Up to $16,000, a single relatively cheap and narrow switch is used. For $32,000 and $64,000, an FNN of two-level trees using the

Key for Network Design Solution Space Summaries (Fig. 1–8)

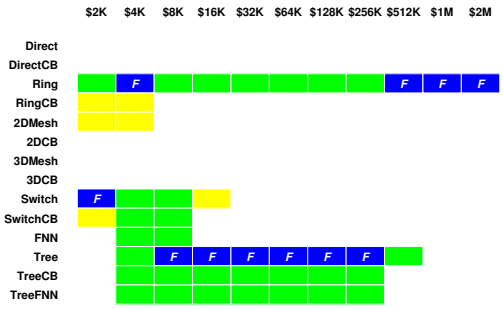


Fig. 1: No Network Performance Constraints

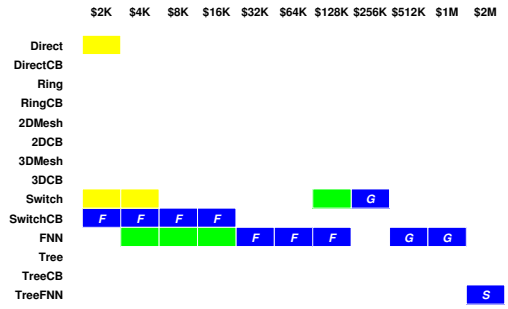


Fig. 2: Max. 50us Latency, Min. 200Mb/s Bisection/PE

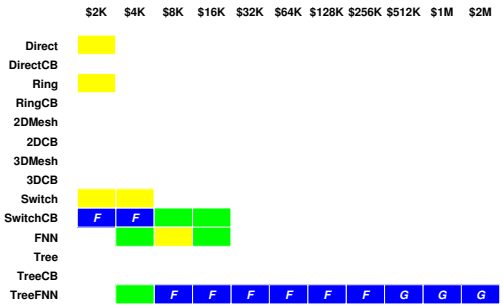


Fig 3: Any Latency, Min. 200Mb/s Bisection/PE

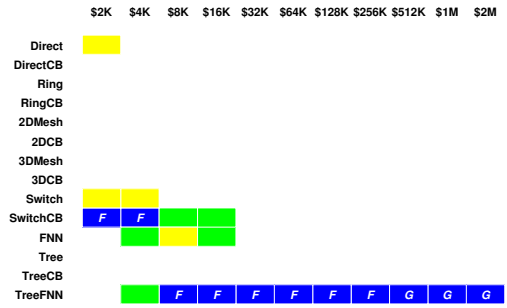


Fig. 4: Max. 100us Latency, Min. 200Mb/s Bisection/PE

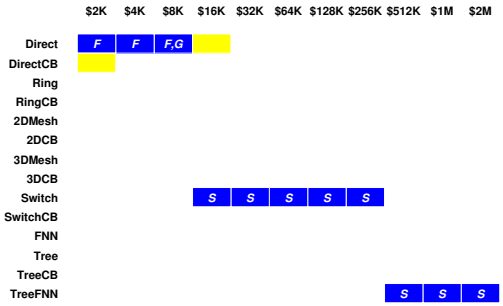


Fig. 5: Max. 10us Latency, Min. 200Mb/s Bisection/PE

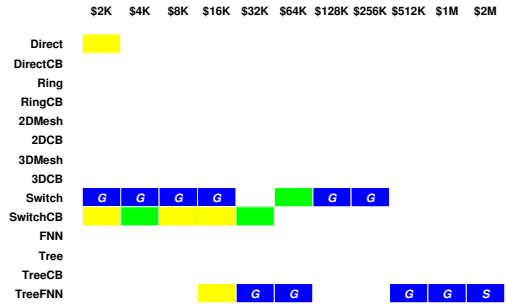


Fig. 6: Max. 50us Latency, Min. 1Gb/s Bisection/PE

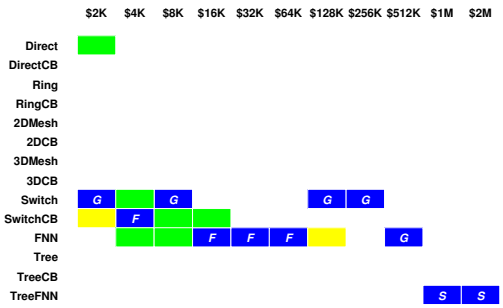


Fig. 7: Max. 50us Latency, Min. 500Mb/s Bisection/PE

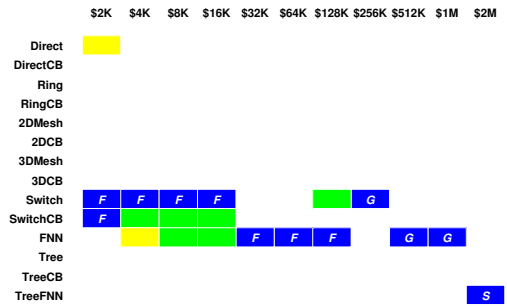


Fig. 8: Max. 50us Latency, Min. 100Mb/s Bisection/PE

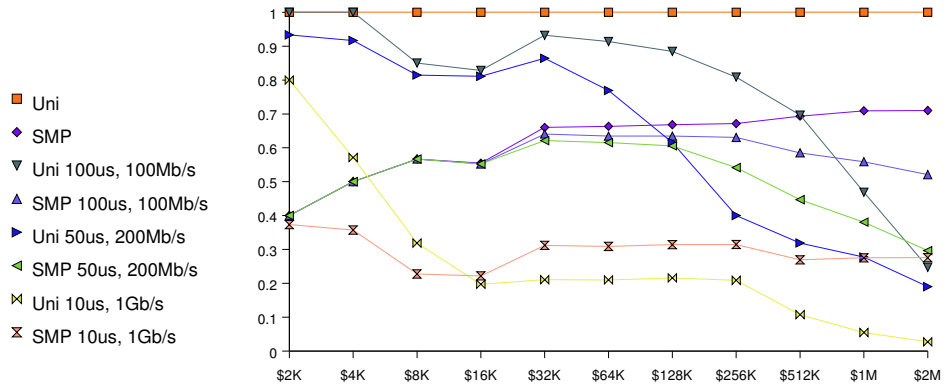


Figure 1: Comparison of uniprocessor and multiprocessor performance as a function of cluster cost for Intel processors with various network latency and bandwidth requirements.

same switches selected for the cheaper systems is employed. The winning $128,000 and $256,000 designs use a single, wide and expensive, Gigabit Ethernet switch. The $512,000 and $1,024,000 winning designs repeat the earlier pattern, reusing the switches from the previous price points in an FNN of two-level trees. Despite the fact that SAN hardware can more easily provide the required bandwidth, the higher cost of SAN hardware only yields the best design at the $2,048,000 price point.

Dropping the bandwidth requirement to 500Mb/s bisection per processor yields the very complex design space summarized in Figure 7. Fast Ethernet re-enters the mix, actually yielding the best designs for four of the price points. However, Gigabit and Fast Ethernets offer comparably good solutions for many of the lower price points. The biggest surprise is that SAN hardware starts to win at $1,024,000 – a lower price point than when 1Gb/s bisection bandwidth per processor was required. This is because the narrower SAN switches are relatively efficiently able to cover the requirement in a two-level FNN topology.

The design space changes only in minor ways as the bandwidth requirement is dropped to 200Mb/s (Figure 2) and even 100Mb/s (Figure 8). Fast Ethernet wins all of the low cost designs and Gigabit Ethernet reclaims the top position for $1,024,000.

3.3 When are Multiprocessor Nodes Better Than Uniprocessor Nodes?

Aside from network issues, one of the most contentious issues in cluster supercomputing involves the use of uniprocessor versus multiprocessor nodes. Intuitively, shared memory multiprocessors seem like a highly desirable node architecture because they directly provide a relatively high-performance type of parallel processing. There also is the classical argument that multiprocessors give clusters more processors with less additional hardware, thus saving on everything from floorspace to interconnection network complexity. Many people find this logic so compelling that they claim there is never any reason to use uniprocessor nodes. On the other hand, many price/performance records have been set by clusters using uniprocessor nodes. Certainly, multiprocessor nodes do not seem to benefit as greatly from commodity pricing as uniprocessor nodes do; often, a two-processor node costs significantly more than two comparably-equipped uniprocessor nodes. Another obvious attribute favoring uniprocessor nodes is the fact that many small multiprocessors divide performance of each subsystem among the processors within a node, yielding poorer performance per processor. This effect is most obvious for performance of memory and network interfaces; however, memory performance is no longer compromised in some multiprocessors, and it also can be argued that many well-tuned cluster application codes are relatively cache-friendly, making main memory performance differences less of a concern. It would be nice to know what the primary engineering tradeoffs really are.

Fortunately, the CDR evaluations are immune to argumentative persuasion. However, some care must be taken to construct experiments so that the multiprocessor versus uniprocessor choice is not convolved with other system effects. It would not be very informative, for example, to compare a uniprocessor AMD Sempron with a multiprocessor Intel Itanium2. However, both Intel and AMD market processors that have essentially the same cores available as both uniprocessor and multiprocessor versions. Thus, our experiments compare Intel Pentium 4 uniprocessor nodes with Xeon multiprocessors (Figure 9) and AMD Athlon 64 uniprocessor nodes with Opteron multiprocessors (Figure 10).

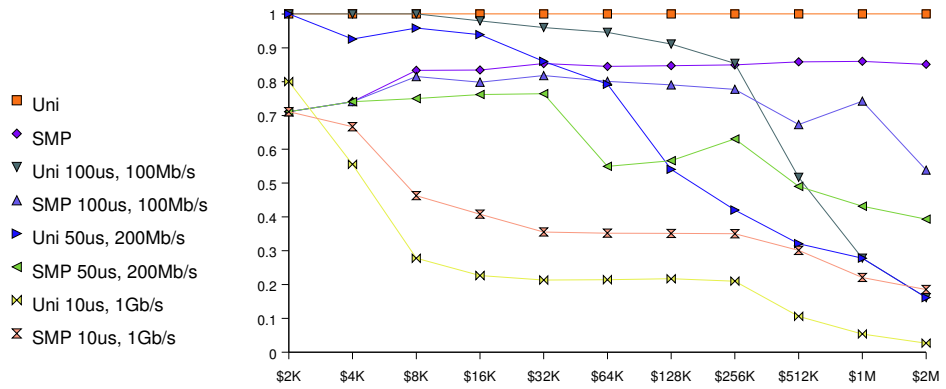


Figure 2: Comparison of uniprocessor and multiprocessor performance as a function of cluster cost for AMD processors with various network latency and bandwidth requirements.

To minimize the impact of memory system differences, the design parameters all assume cache-friendly code, instead focusing on determining how design choices change as the budget is scaled and as network latency and bandwidth requirements become more demanding.

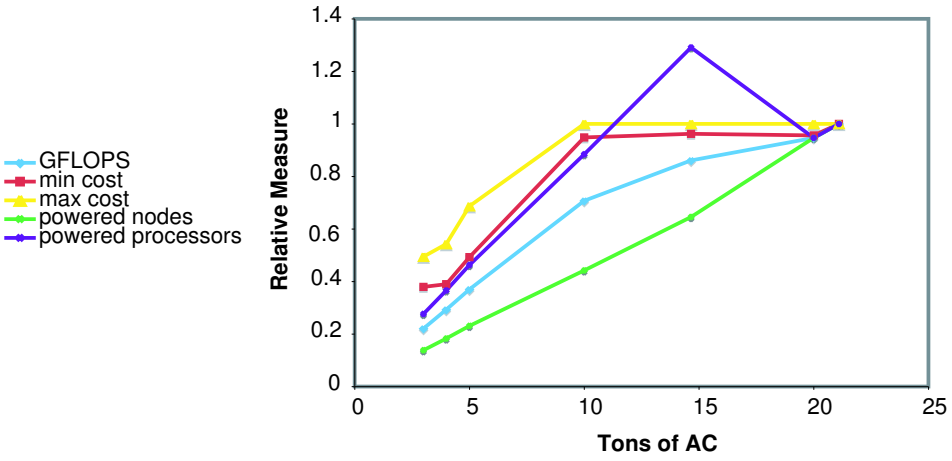
Both Figures 9 and 10 plot the GFLOPS performance of the best uniprocessor and multiprocessor alternatives not as absolute values, but as fractions of the best uniprocessor design's GFLOPS performance for the same cost and no network design constraints (marked "Uni" in both graphs). For example, in Figure 9, the best Xeon (marked "SMP") system for $2,048,000 achieves 71% of the performance obtained by the best Pentium 4 (marked "Uni") system at the same price point. Curves are plotted for the best uniprocessor and multiprocessor systems given network latency and bisection bandwidth per processor design constraints of $100\mu s$and 100Mb/s, $50\mu s$and 200Mb/s, and $10\mu s$and 1Gb/s.

Perhaps the most obvious feature of the plots in Figures 9 and 10 is that requiring very high-performance networks costs enough money to dramatically reduce the potential GFLOPS performance for systems built using either uniprocessor or multiprocessor nodes, but this is fully expected. A somewhat more subtle general trend is that, in comparison to multiprocessor node designs, the uniprocessor node designs are more seriously impacted by high-performance network requirements at the high-end of the price scale. At the low end of the price scale, uniprocessor node designs always outperformed multiprocessor node designs; however, with the exception of the no network design constraints cases, the multiprocessor node systems always performed better than their uniprocessor counterparts at the high end of the price scale. In general, the more demanding the network requirements, the earlier the price point at which multiprocessor nodes start outperforming uniprocessor nodes. However, the actual price points at which crossover occurs vary widely. The design constraints of $100\mu s$and 100Mb/s yielded crossovers between $256,000 and $1,024,000; $50\mu s$and 200Mb/s yielded crossover very close to $128,000 in both the Intel and AMD cases; and $10\mu s$and 1Gb/s yielded crossovers between $4,000 and $16,000. The bigger your budget and the more aggressive your network requirements, the better you should like multiprocessor nodes... but being able to live with a low-performance network using uniprocessor nodes clearly offers the highest GFLOPS per dollar.

3.4 How do Power and Cooling Issues Affect the Design Space?

In addition to various computational performance issues, the CDR models relevant physical properties. Among the physical properties tracked are floorspace, power consumption, and cooling requirements. Power consumption and cooling requirements are very directly related to each other, because the energy that goes into the cluster is nearly 100% converted into heat. The interesting thing about floorspace and power constraints is their relative independence from the other aspects of system design; these constraints are largely orthogonal to other design issues, often acting simply to prune specific design features from the search space.

Consider designing a cluster to meet the $50\mu s$and 200Mb/s constraints we have used in so many of the above examples. Assume a financial budget of $2,048,000, and use the CDR to optimize the design to achieve the maximum possible GFLOPS as the amount of air conditioning available is varied. The results as available air conditioning is increased from 2.5 to 22 Tons are shown in Figure 11. As one would expect, GFLOPS increase as cooling increases, but the rate of increase slows. For relatively low amounts of cooling, it is not possible to productively spend the entire



budget, but there soon comes a point where essentially all the budget is being spent; this can be seen by the flattening-out of both the minimum and maximum costs for viable designs. In this case, the number of powered nodes steadily increases as more cooling is made available, but the number of powered processors builds to a peak around 15 Tons, drops, and then increases slightly again for the case with the most air conditioning available. This strange peak is not unusual; which processors yield the fastest system easily can change depending on "packaging" issues. In this case, the powered processors peak reflected Opteron-based nodes, which were replaced by Xeon-based nodes as cooling capacity increased further.

Overall, it is very difficult to make useful generalizations about power-related performance... which is why it is so important for a tool like the CDR to include power modeling.

4 Related Work

The parallel computer architecture literature does an excellent job of describing how to design parallel computers from scratch [12, 19, 20]. In practice, custom supercomputer designers rely on designs from previous supercomputers and existing parts to reduce development time and cost. Standardized interfaces are the key that lets Beowulf clusters take this approach farther by using only commodity parts. Economies of scale and competition among vendors drive down prices of these interchangeable parts, while driving performance up. If one vendor raises prices or lags in performance, standardized interfaces allow a cluster designer to easily replace the slow or expensive product with a faster or less expensive one. Moreover, standardized interfaces allow vendors of custom hardware to extend the commodity platform in new ways without having to develop an entirely new system. For example, a system area network (SAN) vendor can design a faster cluster interconnect without having to support design of a new CPU, motherboard, memory architecture, etc.

Much has been written about how to construct, manage, and use Beowulf clusters [2, 6, 17, 18, 23, 24, 25]. This documentation is an invaluable source of information on the mechanics of setting up the hardware and systems software of Beowulf clusters and porting applications to run on them. When it comes to the issue of designing the architecture of a Beowulf cluster, however, very little has been written on how to get the best performance for a particular application, while taking price into account. The volatility of the commodity parts market makes it difficult for printed literature to say anything more specific than "it depends" when discussing price/performance tradeoffs. The CDR combines traditional performance analysis with a price database. It methodically searches through millions of combinations of network architectures and parts in its database until it finds the combination that best suits the parameters it is given.

The field of performance modeling and prediction is based on predicting system performance based on architectural features of the entire system. Performance models range from cycle accurate simulations of individual processors and network links to higher-level analytical modeling, based on parameters like individual processor speed, network bandwidth, and network latency. Analytical models, which can be evaluated more quickly than cycle accurate simulations, have been successfully used to predict system performance [4, 5, 28, 11, 14, 21, 26, 27, 29]. Parameters for such models can be collected either through manual analysis of the application program or with tools designed

for collecting them[9, 13, 15, 22, 27]. These tools and analysis techniques can be used to derive the requirements parameters input into the CDR.

Price/performance modeling has been done for commodity clusters based on models for specific applications[1]. However, the modeling was limited to only a few component variations. The CDR handles a much wider design space, and automates its exploration.

The CDR is not the only tool available to cluster designers. Aspen Systems' "Cluster Floorplanning and Power Estimator" (`www.aspsys.com/clusters/estimator/`) addresses space, power, and cooling issues, but their tool does not take performance into account. A least one vendor has a "cluster design" web form giving pricing information to potential customers who specify system parameters, like processor type, number of processors, amount of memory per node, network type, etc (`www.creativec.com/cluster_design.htm`). Using these vendor services is easier than pricing individual components with online pricing services, and they allow the buyer to be aware of volume discounts. However, they do not give any insight into which components a cluster designer should use. In contrast, the CDR uses higher level information about the type of applications that will run on the cluster and searches for system parameters that best meet the application's needs.

5 Conclusion

The availability of low-cost interchangeable parts has put a lot of people in the position of designing a custom supercomputer for their own use. Though traditional design methods still apply, the large number of choices of parts and frequently changing prices make rules of thumb difficult to apply effectively. The CDR helps cluster designers tailor their supercomputer to a particular application set's requirements. Although the user inputs are inherently approximate, as they must be to keep the volume of input data reasonable, the CDR has proven to be a useful tool for hundreds of users over the past several syears.

We expect to continue to evolve and improve its ability to model system performance and cost. To that end, we have made source code available under the BDR (Beowulf Design Rules) project at SourceForge.net to allow others to modify and experiment with the CDR.

References

[1] J. C. Becker, B. Nitzberg, R. F. Van Der Wijngaart, and M. Yarrow. Predicting price/performance trade-offs for whitney: A commodity computing cluster. *The Journal of Supercomputing*, 13(3):303–319, May 1999.

[2] Robert G. Brown. Engineering a beowulf-style compute cluster. http://www.phy.duke.edu/brahma/Resources/beowulf_book/.

[3] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceedings of SC2000 Conference on High Performance Networking and Computing*, November 2000.

[4] M. J. Clement and M. J. Quinn. Analytical performance prediction on multicomputers. In *Supercomputing '93: Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 886–894, New York, NY, USA, 1993. ACM Press.

[5] M. J. Clement and M. J. Quinn. Architectural scaling and analytical performance prediction. *International Journal of Computer Systems Science and Engineering*, 11(3):159–167, May 1996.

[6] Henry G. Dietz. Linux parallel processing HOWTO. http://aggregate.org/PPLINUX/.

[7] Henry G. Dietz and Tim I. Mattox. KLAT2's flat neighborhood network. In *Proceedings of the Extreme Linux track of the 4th Annual Linux Showcase*, Atlanta, GA, USA, October 2000.

[8] Henry G. Dietz and Timothy I. Mattox. Inside the KLAT2 supercomputer: The flat neighborhood network & 3DNow! http://arstechnica.com/cpu/2q00/klat2/klat2-1.html.

[9] T. Fahringer, A. Pozgaj, J. Luitz, and H. Moritsch. Evaluation of p3t+: A performance estimator for distributed and parallel programs. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Cancun, Mexico, May 2002.

[10] Thomas Hauser, Timothy I. Mattox, Raymonnd P. LeBeau, Henry G. Dietz, and P. George Huang. High-cost CFD on a low-cost cluster. In *Proceedings of SC2000 Conference on High Performance Networking and Computing*, November 2000.

[11] Adolfy Hoisie, Olaf Lubeck, Harvey Wasserman, Fabrizio Petrini, and Hank Alme. A general predictive performance model for wavefront algorithms on clusters of smps. In *The Proceedings of the International Conference on Parallel Processing*, Toronto, Canada, 2000.

[12] Kai Hwang. *Advanced Computer Architecture*. McGraw-Hill, Inc., 1993.

[13] Darren J. Kerbyson, Hank J. Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J. Wasserman, and Michael Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of SC2001*, 2001.

[14] Darren J. Kerbyson, Harvey J. Wasserman, and Adolfy Hoisie. Exploring advanced architectures using performance prediction. In *Proceedings of the International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA)*, page 27, Kohala Coast, Big Island, Hawaii, January 2002.

[15] Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In *SIGMETRICS 2004/PERFORMANCE 2004: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 2–13, New York, NY, USA, 2004. ACM Press. Predictions of L1, L2, TLB & execution time for an Origin 2000 on several benchmark codes. Based on characterizing binaries in terms of arch-neutral computation and memory access models.

[16] Timothy I. Mattox, Henry G. Dietz, and William R. Dieter. Sparse Flat Neighborhood Networks (SFNNs): Scalable Guaranteed Pairwise Bandwidth and Unit Latency. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005), Workshop on Massively Parallel Processing (WMPP)*, April 2005.

[17] B. McGarvey, R. Cicconetti, N. Bushyager, E. Dalton, and M. Tentzeris. Beowulf cluster design for scientific PDE models. In *Proceedings of the 2001 Annual Linux Showcase*, Oakland, CA, November 2001.

[18] Jacek Radajewski and Douglas Eadline. Beowulf HOWTO. http://en.tldp.org/HOWTO/Beowulf-HOWTO.html.

[19] Howard Jay Siegel. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*. McGraw-Hill, New York, second edition edition, 1990.

[20] Burton Smith. The end of architecture. *ACM SIGARCH Computer Architecture News*, 18(4):10–17, 1990.

[21] Allan Snavely, Laura Carrington, Nicole Wolter, Jesus Labart, Rosa Badia, and Avi Purkayastha. A framework for application performance modeling and prediction. In *Proceedings of the ACM/IEEE Supercomputing Conference*, pages 24–40, Baltimore, MD, November 2002.

[22] Michael R. Steed and Mark J. Clement. Performance prediction of pvm programs. In *International Parallel and Distributed Processing Symposium*, 1996.

[23] Thomas Sterling. *Beowulf Cluster Computing with Linux*. MIT Press, 2001.

[24] Thomas Sterling. *Beowulf Cluster Computing with Windows*. MIT Press, 2001.

[25] Thomas Sterling, John Salmon, Donald J. Becker, and Daniel F. Savarese. *How to Build a Beowulf*. MIT Press, May 1999.

[26] Valerie Taylor, Xingfu Wu, Jonathan Geisler, , and Rick Stevens. Using kernel couplings to predict parallel application performance. In *Proceedings of the IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, Edinburgh, Scotland, July 2002.

[27] Dvorak Vaclav and Cejka Rudolf. Performance prediction of cluster computing. In *Proceedings of the International Workshop Modelling and Simulation in Management, Informatics and Control MOSMIC'99*, pages 229–233, Sulov, SK, 1999.

[28] A. van Gemund. Performance modeling with pamela: An introduction, 1992. Describes Pamela (PerformAnce ModEling LAnguage) and a language-based approach to describing program and machine characteristics. Specifies how delays for communications, etc., affect critical paths, etc.; "serialization analysis.".

[29] Xingfu Wu, Valerie Taylor, Jonathan Geisler, and Rick Stevens. Isocoupling: Reusing coupling values to predict parallel application performance. In *The International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, April 2004.