Locality of Distributed Architectures and Algorithms

by:

Kevin J. Butler

December 13, 1995

submitted to:

Mark Clement, Ph.D.

CS 501R

Table of Contents

Abstract

The principle of locality of reference can be used to analyze suitability of algorithms for implementation on specific distributed computer architectures. In traditional computer architectures, the type of locality exhibited by an algorithm determines the effectiveness of different code and data cache architectures. Similarly, the type of locality exhibited by a distributed algorithm determines its suitability for implementation on a particular distributed architecture. A technique to graphically represent the locality characteristics of distributed architectures and distributed algorithms is presented, and an implementation of the technique is described.

Introduction

In a doctoral dissertation written at Brigham Young University (Grimsrud, Knut S., Quantifying Locality, Brigham Young University, 1993), Knut Grimsrud categorized memory access locality according to various program constructs, including sequential code execution, loops, array data access, and record data access. Caches which are optimized for a certain type of locality do not perform well for processes which exhibit other types of locality. For example, an instruction cache often includes a prefetch cache to optimize sequential instruction fetches. Data accesses generally do not follow the sequential access pattern, and thus using a prefetch cache for data caching results in little improvement or even a reduction in performance. Thus, knowledge of locality characteristics can be used in designing traditional computer architectures to enhance performance.

Distributed algorithms exhibit a similar locality of reference. Each process in a distributed machine has access to a subset of the information in the problem. In the case of Uniform Memory Access (UMA) architectures, each process has equal access to all the data. In Non-Uniform Memory Access (NUMA) machines, each process has fast access to only a proper subset of the information, and if other information is required, the process must obtain it from another process via a slower communication path.

By graphically representing the locality characteristics of a distributed algorithm and a distributed computer, suitability of the architecture for implementation of the algorithm can be determined. This paper describes the locality factors which affect performance of distributed algorithms, shows examples of the graphical representation of these characteristics for one host architecture and three algorithms, and describes the software used to develop these representations.

Locality for Distributed Architectures and Algorithms

Speed of computation and speed of communication determine the effectiveness of a distributed computer for hosting various applications. Similarly, an algorithm’s requirements for communication (including latency and bandwidth) and for computation determine the distributed architectures to which it is most suited.

In order to evalutate architectures for a given algorithm (or vice versa), it is helpful to have a tool which allows comparison of features of the architecture with corresponding requirements of the application. For the host architecture, it is useful to display communication delays as a function of both message size and the speed of computation. For the algorithm, the corresponding factors are the number of messages during a run or set of runs of the algorithm as a function of message size and the grain size, or computation time between communications.

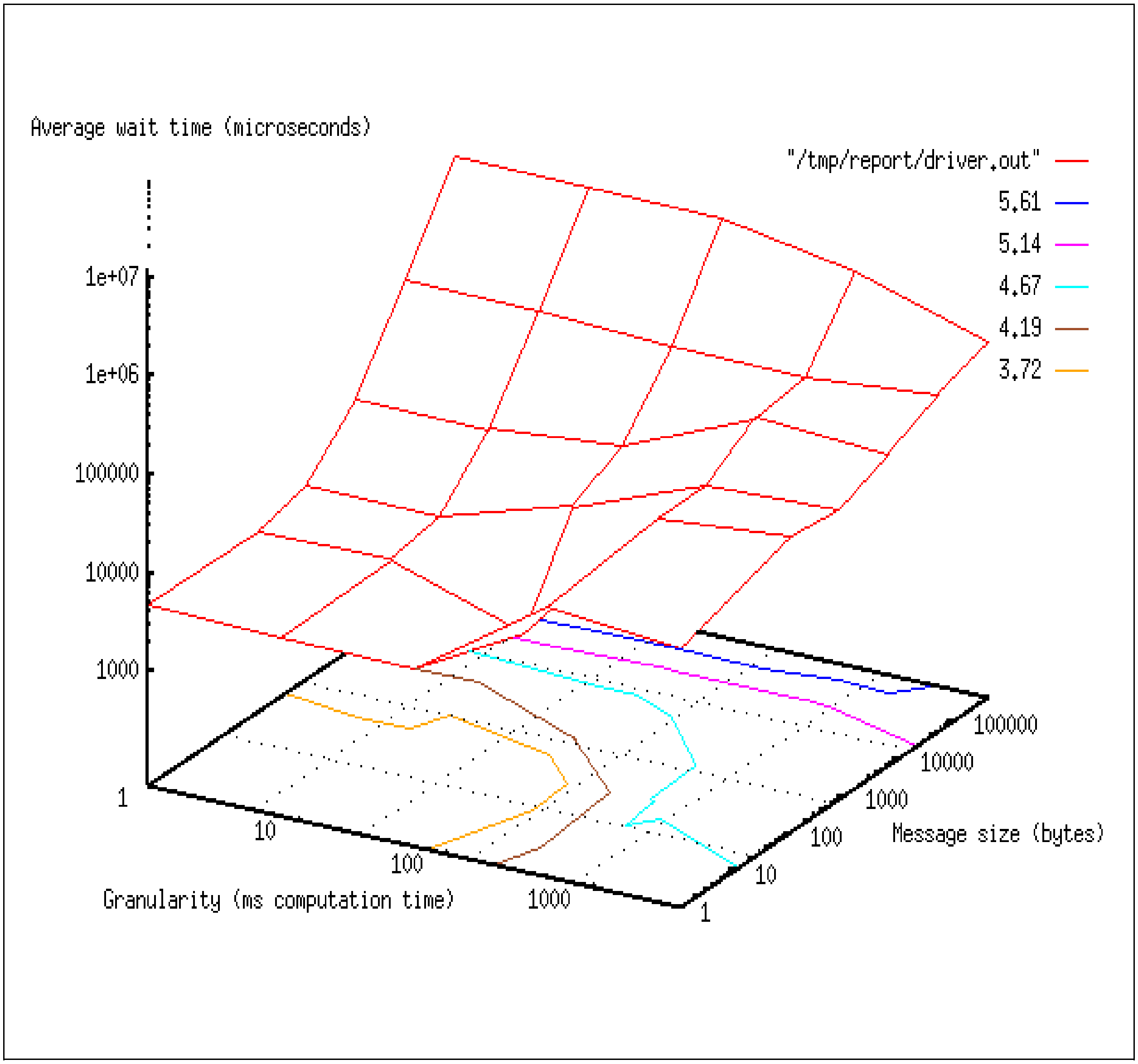

Figure 1PVM Driver – Performance of a Network Multicomputer. This figure shows the wait time (Z a round trip message of different message sizes (Y axis), and illustrates the possible performance overlapping communication and computation (X axis). Note the performance gain for large messag very large grain size (the upper right) as computation and communication are overlapped very effe

Graphical Representation of Locality Characteristics

Architecture

Figure 1 shows the locality characteristics of an ethernet network of workstations communicating via message passing over PVM. The time (in microseconds) is the round trip time for a message of a given size, minus the time for any computation algorithm could perform while waiting for the message. Generally, small messages require less time, large message require more time.

Some interesting features of the graph include:

- The performance gain for overlapped communication on very large messages(upper right). This is because PVM utilizes asynchronous communication and thus allows the application to continue computing while the network is transmitting the message.
- Little or no performance gain for overlapped communication on small messages (lower side). This is because the majority of the time component for these messages is the PVM packing and unpacking overhead, which is synchronous and therefore not overlapped. Note that this overhead can be considered part of the latency, but that it increases with the message size (and probably with message complexity).
- The sudden increase in time for small messages with high granularity (lower right). This is probably because the return message arrived, but the application did not request it from the communication subsystem, and therefore it was paged out by the virtual memory system, causing a page fault in order to retrieve the message.
- Minimum time for a message is on the order of 10 ms.
- The change from latency-bound communication to bandwidth-bound communcation (near the $10^{4.67}$ microsecond line--at 1000 bytes in the upper left down to 10 bytes at the lower right)

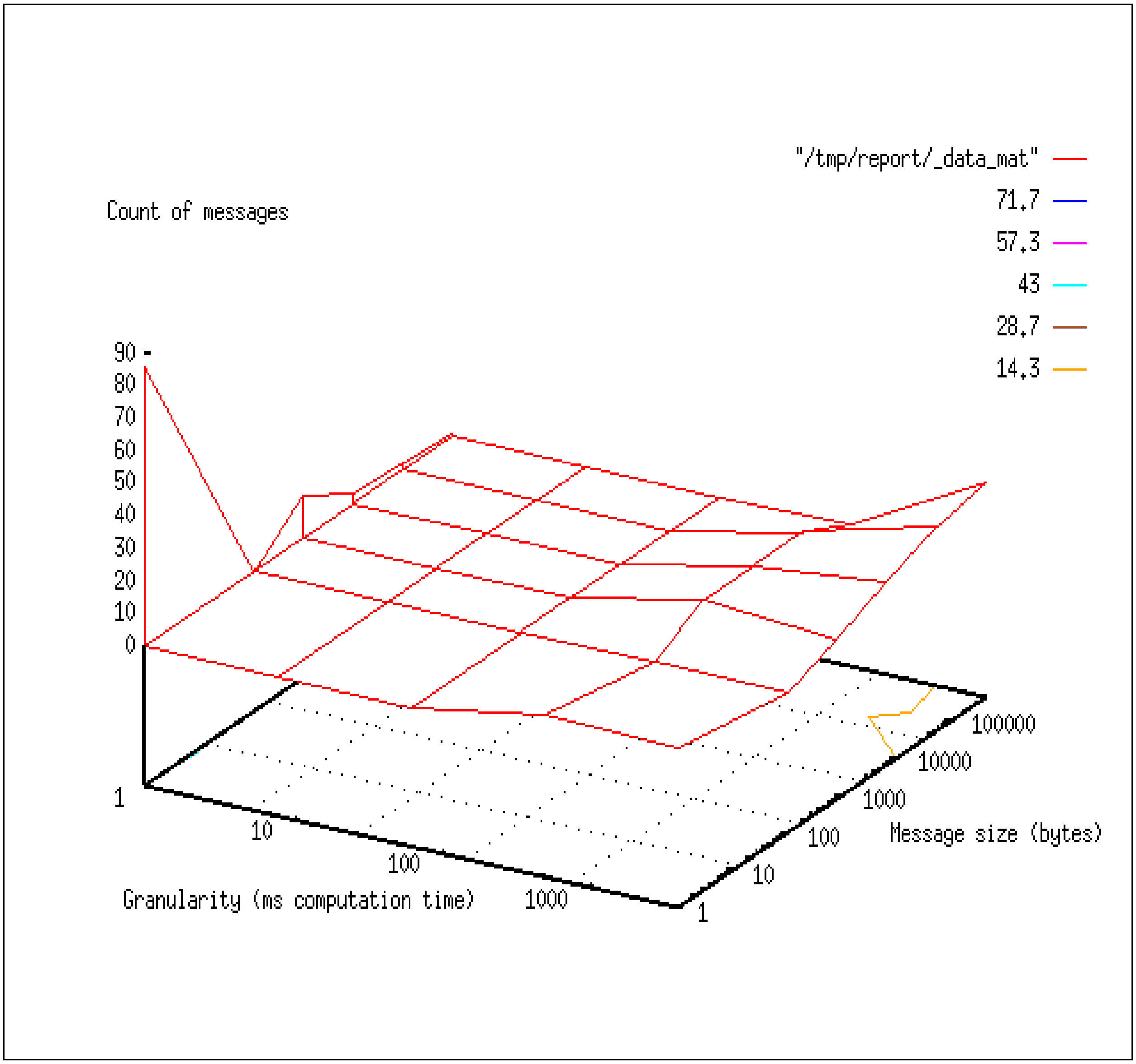

Figure 2. PVM Recorder – Matrix Multiplication cumulative results. Shows number of messages (Z different sizes (Y axis) and the granularity (computation time) between communications (X axis). C results for various problem sizes, numbers of processors, and both the row and block algorithms. N low number of messages and the high amount of computation between messages, showing large g

Algorithms

Matrix Multiplication.

Figure 2 shows the accumulated locality characteristics of several runs of both the row- and block-oriented matrix multiplication algorithms. The Count of messages is the total number of messages of a given size granularity during all the runs. The Message size is the number of bytes in the message (excluding any header information, and the numbers on the axis show upper bound of the message size group--e.g., an 11 byte message is placed in category 100), and the Granularity is the amount of computation time between the current message and the preceding message (again, the numbers represent the upper bound of the categories).

Some interesting features of the graph include:

- The multi-cast silhouettes along the upper left. These result from a multi-cast or multi-receive operation, and thus have virtually no computation between communications.
- Large number of small multi-cast messages--these consist primarily of multi-cast/receive signals--"Start", "Complete", "Acknowledge", etc.
- The low number of messages--the largest category (small multi-cast messages) has only 80 messages accumulated over several different runs.
- Most large messages have a large amount of overlapped computation (upper right versus upper left).

Comparing Figure 2 and Figure 1 shows that matrix multiplication is a good fit for networks of workstations, as there is 1) little communication, and 2) virtually no communication in the large-size, low-overlap area (upper left).

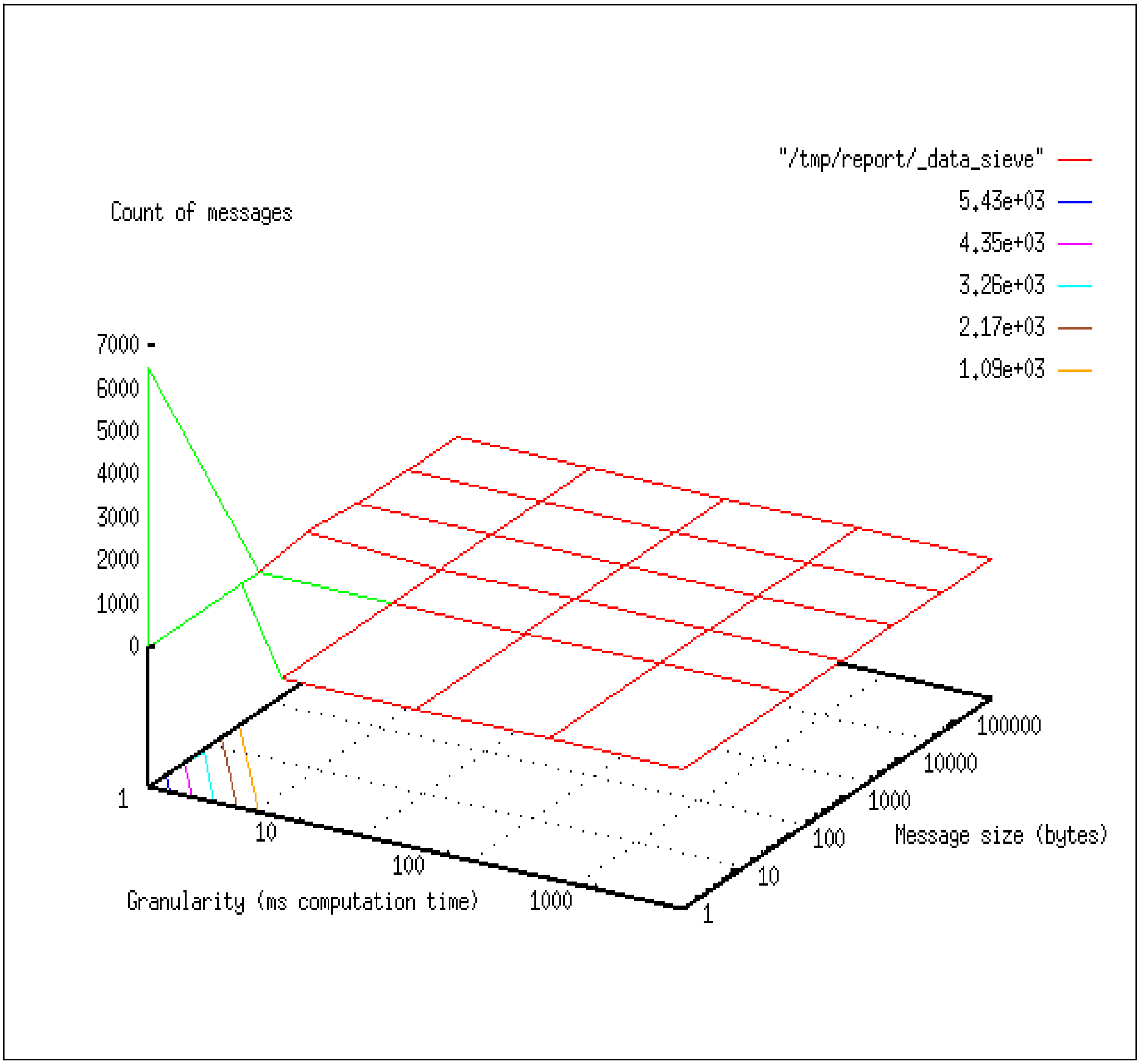

Figure 3. PVM Recorder – Sieve of Eratosthenes cumulative results. Note the large number of sma messages, with little computation between them, illustrating very small grain size.

Sieve of Eratosthenes.
Figure 3 shows the accumulated locality characteristics of several runs of an implementation of the Sieve of Eratosthenes algorithm. Figure 4 shows an enlarged view of the near-zero count portion of the surface.

Some interesting features of the graphs include:

- The large number messages (6000 vs 100 for matrix multiplication).
- The ratio of small messages with little overlapped computation to messages with

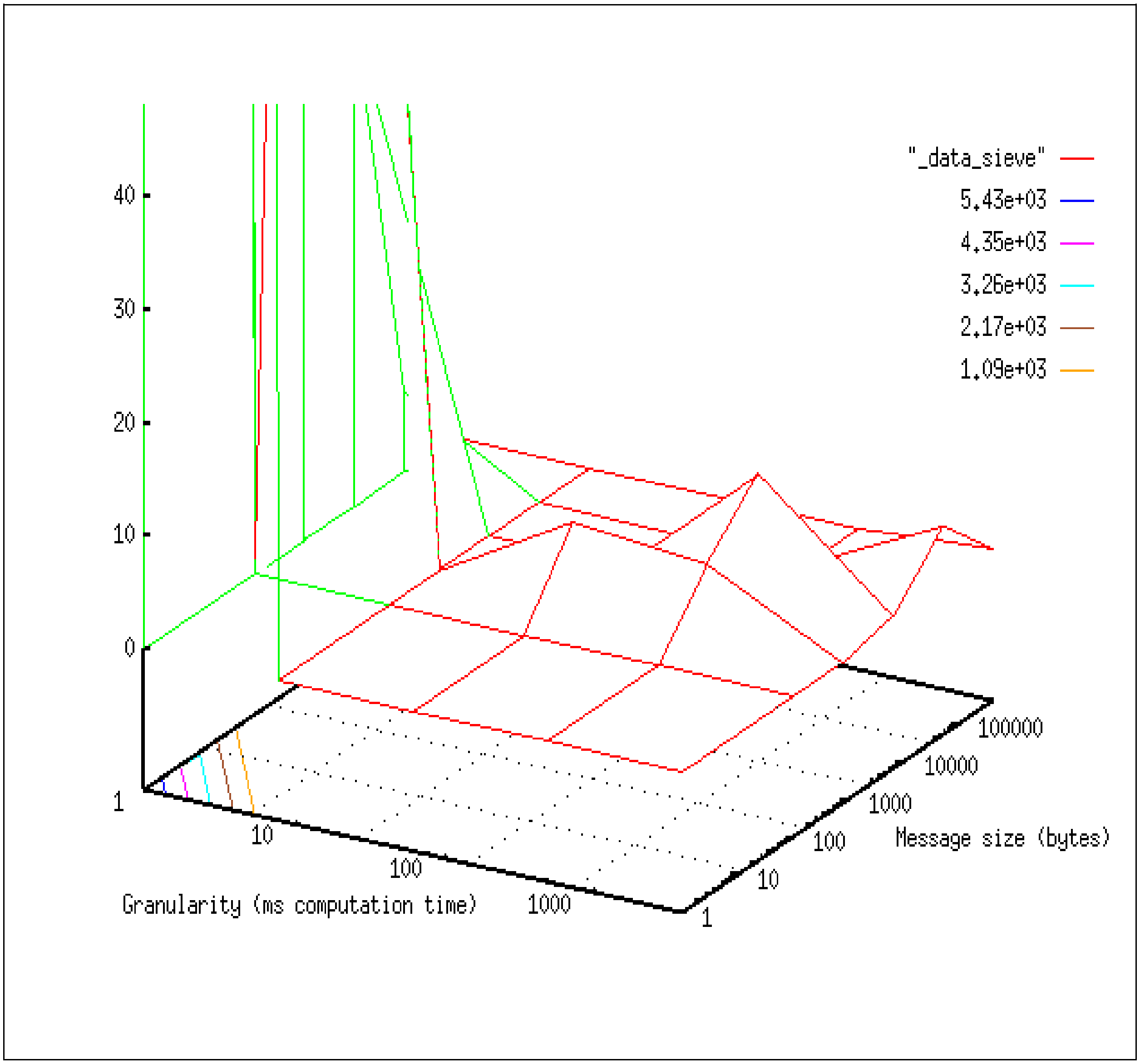

Figure 4. PVM Recorder – Sieve of Eratosthenes cumulative results, enlarged view. This view sho large messages with some computation between messages.

some overlapped compuation. This is because the algorithm primarily exchanges messages containing only a single prime number.

- (Enlarged view) The presence of some messages with larger sizes and more computational overlap. These messages are the gathering of results, which overlaps with the collating of the results.

Comparing these two figures with Figure 1 illustrates that the Sieve of Eratosthenes algorithm is a poor match for a network of workstations. It could be improved, however,

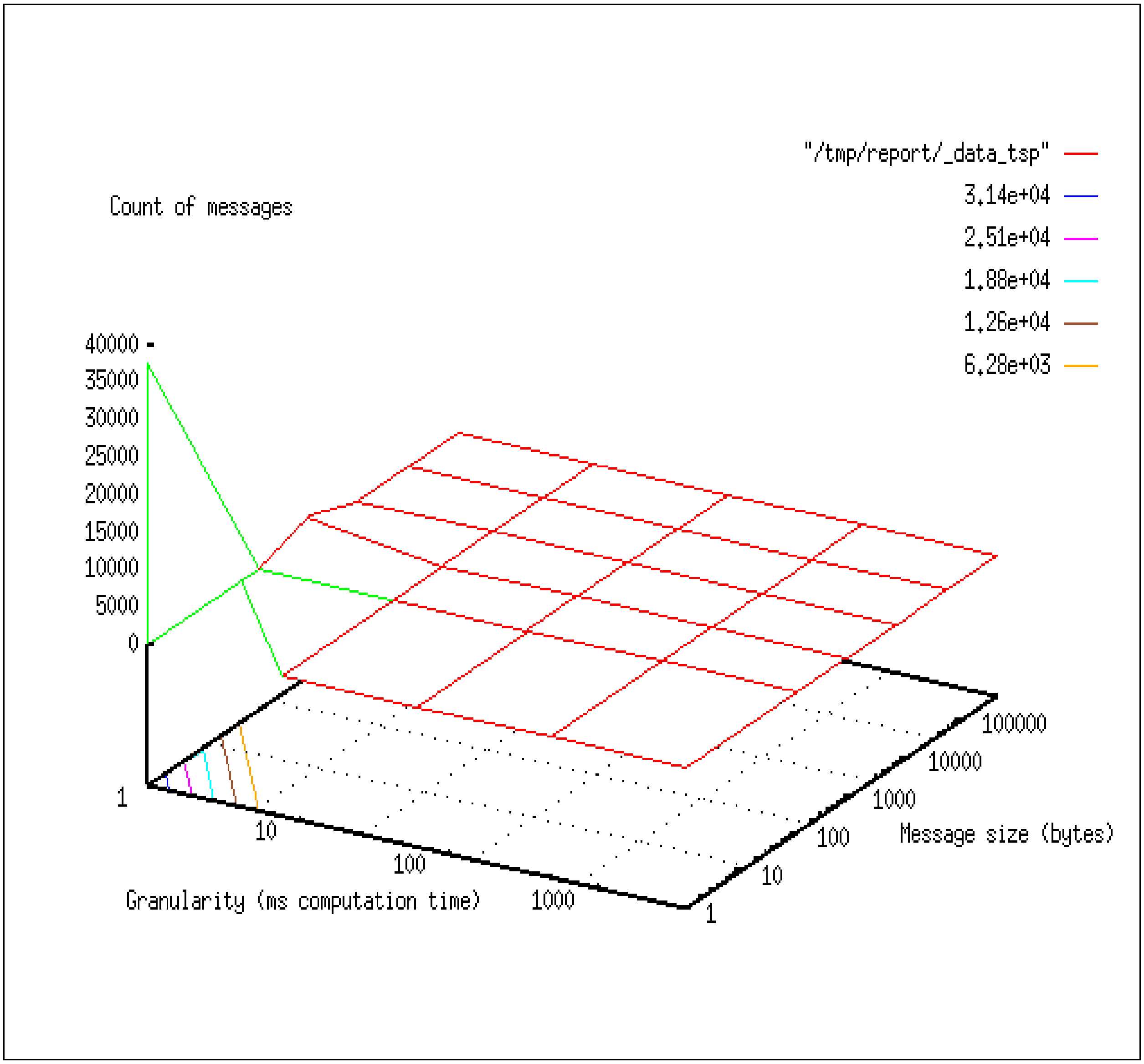

Figure 5. PVM Recorder – Traveling Salesperson cumulative results. Note the strong presence of messages with little computation. This shows a very small grain size.

by reducing the quantity of the small messages--perhaps by sending multiple primes in a single message, rather than a message per prime found.

Traveling Salesman.

Figure 5 shows the accumulated locality characteristics of several runs of a Traveling Salesman application, implemented with the furthest-from-the-tour heuristic. Figure 6 shows an enlarged view of the near-zero count portion of the surface.

Some interesting features of the graphs include:

- Very large number of messages (40,000)
- Complete lack (enlarged view) of overlapped computation.
- Decreasing numbers of messages with increased message size. The larger messages are only startup and results messages, and are thus infrequent.

Comparing these two figures with Figure 1 illustrates that this traveling salesman algorithm is a very poor match for the network of workstations architecture--worse than the Sieve of Eratosthenes. Also, because of the nature of the heuristic chosen, there is

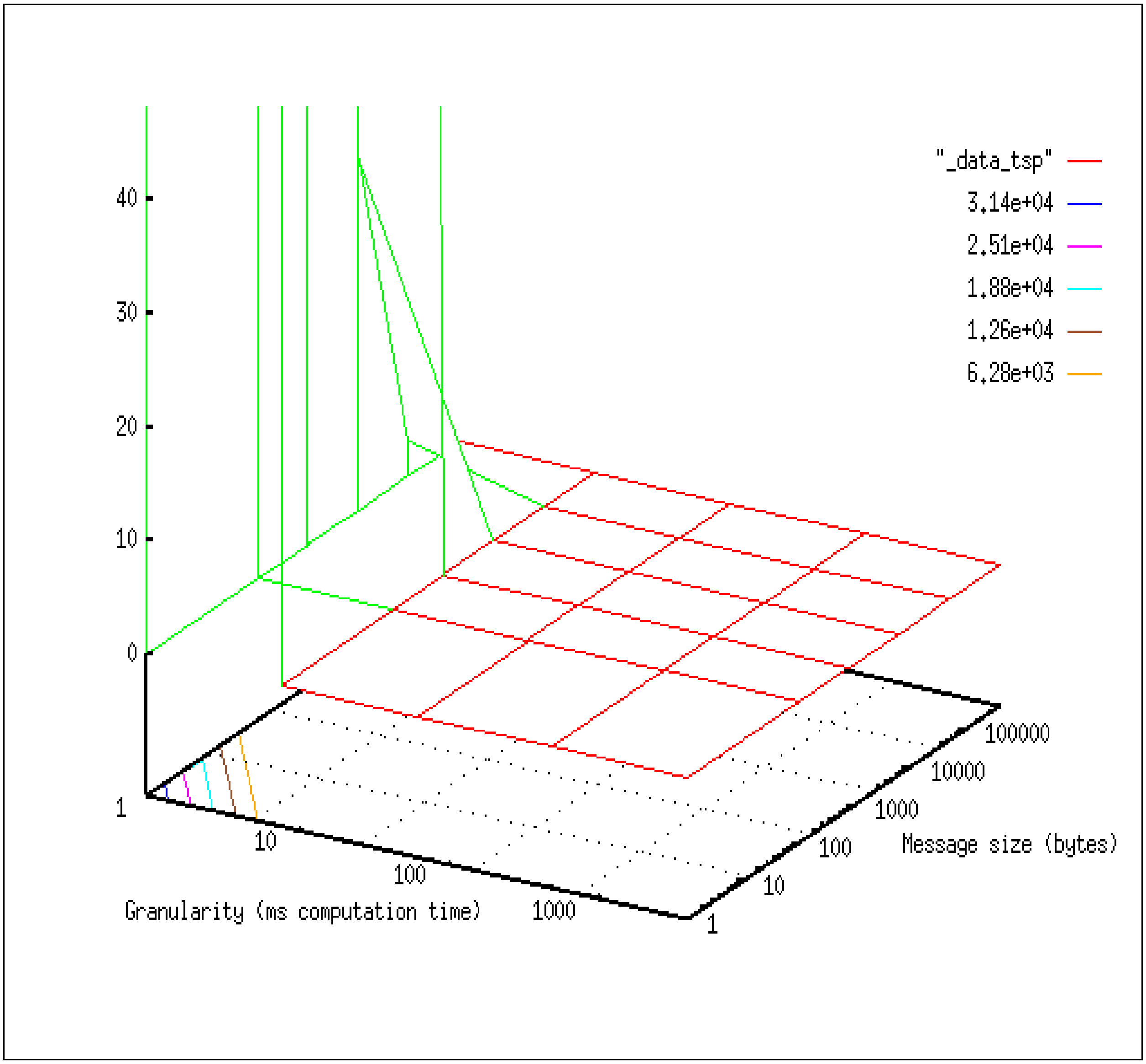

Figure 6. PVM Recorder – Traveling Salesperson cumulative results, enlarged view. In contrast to of Eratosthenes example, enlarging the view of the Traveling Salesperson Performance reveals no grain size" portions of the algorithm.

no way to simply combine several of the messages into a single larger one, because the contents of the messages are dependent upon previous messages.

Implementation

PVM implements a message-passing virtual machine. The PVM API is the de facto standard for portable message-passing applications, and presents a clean division between the application and the communications architecture.

In order to generate the data for the architectural locality characteristics, a PVMDriver application was created which tested the round-trip message time for various sizes of messages and simulated varying amounts of computation during message transport. These test were performed multiple times and the results averaged to reduce the effects of uncontrolled external conditions (such as network traffic).

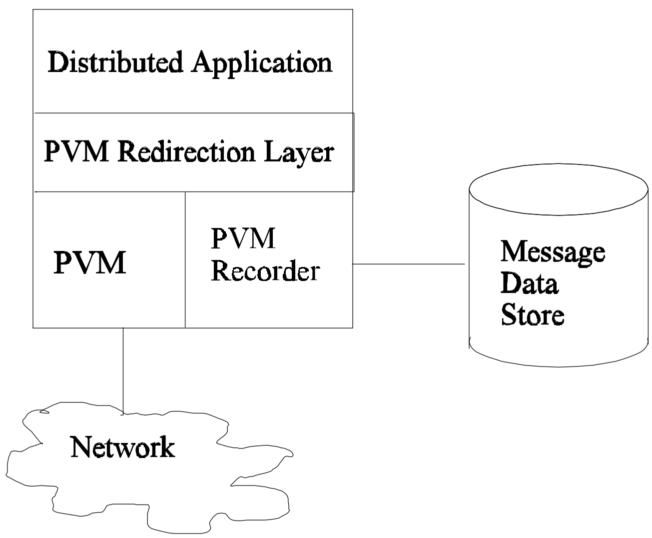

Figure 7 Architecture of PVM Recorder in a distributed application

In order to generate the application data, the PVM API calls were redirected at compile time to an intermediate layer of software, which utilized a PVMRecorder (Figure 7) to record the data about the virtual machine and each message as it is sent or received. This layer is also capable of recording the data of each received message for later playback, though this functionality is not needed for the presented results. Because of this compile time API redirection, a simple recompilation is sufficient to convert an application for data recording.

The output of the PVMRecorder was filtered to obtain count information for each category of message, and then displayed using GnuPlot, which output was captured by XV and converted to an appropriate graphics image format.

Limitations of the PVMDriver and PVMRecorder implementations:

- The gathered data represent only a single node’s picture of the machine. If nodes have asymmetric communication patterns, the current implementation cannot represent the entire picture.

- PVMDriver sends only simple (but possibly large) messages. Times for more complex messages may include more latency from PVM packing and unpacking overhead.
- The PVMDriver times reflect round-trip times. There may exist applications for which this is not accurate.
- PVMDriver busy waits while waiting for a message. This is to prevent loss of the CPU and a scheduler delay to receive the message. However, the scheduler may treat this type of loop differently than a truly CPU-intensive process.

Directions for Future Research

- Examine more tightly-coupled architectures and more algorithms
- Identify other features of the surfaces (like the multi-cast silhouettes) which show various aspects of the algorithm
- Examine other locality characteristics, including distance to sender/receiver of message
- Develop an automated way to transform the recorded data into a presentation format

Conclusion

The locality characteristics surfaces are an effective tool for determining appropriateness of an application for a distributed machine architecture.