Asian Jouranl of Information Technology 5 (2) : 118-125, 2006

Reuse-Based Software Development for Parallel Graphics on Hybrid Architectures

Ali M. Meligy,
Menoufya University, Egypt

Abstract: Performance of programs is often sensitive to the system architecture, the run-time environment and input data characteristics. Therefore, writing portable programs for parallel and distributed systems can be very difficult due to the wide variety of system architectures. To aid parallel program developers, some approaches have been proposed that employ high-level models of parallel computation or different abstraction techniques, such as communication libraries, macros, and abstract data types. A parallel template is a re-usable, application-independent encapsulation of a commonly used parallel pattern. The MIMD-SIMD hybrid system makes use of 'Off-the-Shelf' hardware components. In this study we suggest a template-based methodology to parallel software development on such systems. We use some modified parallel Bresenham's algorithms to illustrate our approach. We also extend some previous results concerning the parallel line-drawing algorithms. These algorithms can be implemented as a re-usable code-skeleton for quick and reliable development of parallel applications. We give a standard interface which facilitates the composition of templates and provide a systematic development model for the hierarchical development and the subsequent requirements of a vast majority of parallel applications, which can be suitably solved on a hybrid system.

Key words: Hybrid architectures, parallel computer graphics, software reusability, systola 1024, MIMD-SIMD systems, parallel beresenham's algorithms

INTRODUCTION

To solve new problems, software developers can rapidly construct applications by assembling reusable components. This increases productivity by avoiding development and improves the quality of software systems by incorporating components whose reliability has already been established. Many research approaches contribute to advances in reuse-based software development. These include creation of new reuse frameworks, processes, architectures, tools and environments as well as formulation of new reuse ideas and concepts[1]. On the other hand, the development of truly reusable parallel libraries is difficult because of additional complexities associated with concurrency and data distribution. An unfortunate consequence of concurrency is that two processes or functions that execute correctly in isolation may not execute correctly when composed, because of race conditions. Data-distribution issues can lead to both correctness and performance problems. If a function expects data to be distributed in one fashion and receives it in another, then either the function will execute incorrectly (in the worst case) or an expensive redistribution operation may be required[2].

The growth in the usage of Internet forces us to re-evaluate CBR. Today it is possible to produce a reusable component that runs in any kind of platform. It is also possible to download these reusable components from Internet and assemble them. It can be expected that in a very near future many academic, governmental and commercial organizations will be marketing the components that they produced on Internet. Therefore, Internet with the components containing analysis models, design patterns, documentations etc. will become the component repository of any organization. Such an environment with thousands of component servers requires reuse support tools which can handle the dynamic changes in the components or component servers. Therefore, Software reuse can be addressed at three levels[3]:

- The techniques for developing high-quality software components.
- An organization that has the resources for producing, consuming and managing shared reusable components.
- Performance evaluation tools in such an organization.

The application of a program-generation approach to enable the reuse of software in an area of scientific computing has been described[4]. The authors describe the design of a program generator for the specification of subroutines that can be generic in the dimensions of

Corresponding Author: Ali M. Meligy, Menoufya University, Egypt

arrays, parameter lists and called subroutines. They describe the application of that approach to a real-world problem in scientific computing, which requires the generic description of inverse ocean modeling tools. In addition to a compiler that can transform generic specifications into efficient Fortran code for models, they have also developed a type system that can identify possible errors already in the specifications. This type system is important for the acceptance of the program generator among scientists because it prevents a large class of errors in the generated code.

The development of efficient parallel programs requires deep knowledge of parallel algorithms and expertise in the application domain. In[5] a method has been presented to make such expertise available in an domain specific tool set. Its construction is based on extensive use of a variety of powerful reuse methods. It automates a large amount of the software construction process, such that users need not know about parallelism.

The term "Template" has recently gained attention in the parallel processing community, yet is an evolving concept with several definitions. The primary motivation for constructing templates is to rapidly infuse into common usage state-of-the-art algorithms in a form which can be adapted to specific application requirements. This implies that the template retains the desired properties but is cast in a form which is independent of parallel architecture, data layout and programming language. Templates go beyond pseudo-code to become objects which are directly compilable on multiple architectures.

The complexity of building parallel applications is the maximum at the lowest level of abstraction e.g. at the socket level which is closest to the hardware architecture. The complexity decreases with higher levels of software abstraction. The levels of abstraction move the user farther from the architectural details. The cost is some performance overhead. A majority of the approaches, proposed to aid parallel program developers, employ a high level model of parallel computation, thus hiding details of low-level parallelism related issues, such as: hardware architectures, interconnection topologies, process creation and binding communication and synchronization, etc. Different models employ different abstraction techniques, such as communication libraries, macros, new parallel languages and abstract data types.

Depending on the degree to which the programmer specifies the parallel interactions, these models can broadly be categorized as:

- **Explicit:** Message Passing Library (MPL) packages like PVM[6] and MPI[7] and various Remote Procedure Call (RPC) packages fall into the explicit category.
- **Implicit:** Techniques like parallelizing compilers[8] fall into the implicit category. Many functional and logic programming languages also explore implicit parallelism which is naturally present in these languages.
- **Semi-explicit:** Here the user handles part of the parallelism-related issues. For instance: object-oriented programming models based on active objects and various template-based[9].

Template-based approach for parallel programming is based on the use of frequently occurring patterns for parallelism. Various terminologies have been used in the literature to express similar ideas, but in different contexts. For example, the term design pattern is used in[10] for representing frequently occurring patterns in the context of object-oriented design methodology. The same terminology has been used in the context of parallel and distributed computing to imply commonly occurring parallel or distributed computing abstractions[11].

The parallel template used here is different from a C++ template. A template-based approach has the same intention as the other approaches to parallel programming, i.e. to facilitate the development of a parallel application. In addition, it emphasizes the following issues: (a) re-usability of the frequently occurring structures for parallelism and of the existing sequential code; (b) usability through the study and documentation of the frequently occurring patterns for parallelism, thus avoiding re-learning solution strategies; and (c) the possibility of writing correct programs with lesser effort by hiding most of the tedious and error-prone, low-level parallelism related details from the user.

In[12] a programming paradigm for parallel computing is defined as a class of algorithms that solve different problems but have the same control structure. The term Archetype is used in[13] to denote a program design strategy for a class of parallel algorithms along with the associated program design and example implementations. Emphasis here is on enhancing a developer's understanding of common classes of parallel problems using documentation and exemplar implementations.

Some characteristics of a template-based model are: Separation of specification, hierarchical resolution of parallelism, mutually independent templates, extendible repertoire of templates, large collection of useful templates, open system concept, correctness, use of an existing programming language rather than language extension or a new language which will enhance re-usability of existing sequential code and also reduce learning and development times, performance, support tools, portability, flexibility, and usability[9].

In the field of computer graphics, images are produced using primitive graphic objects such as points, straight lines and circles. Computers with a single processor may take several hours to generate complex images. This is not acceptable, especially for representing dynamic phenomena. Higher performance can be achieved by using multiprocessor machines that provide for concurrent execution of programs. In general, the line segment is the basic entity in all computer graphics systems. Straight-line segments exist in block diagrams, bar graphs, drafts for civil and mechanical engineering, architectural plans and logic diagrams. Curves can be approximated using a series of small line segments. In graphic interactive applications, the problem is how to obtain pixels that provide the best approximation to a line in a shorter time.

Flynn classifies parallel machines in terms of the number of distinct instructions issued at a time and the number of data elements they operate on. The conventional Von Neumann sequential computer is a SISD (single-instruction single-data). Parallel machines are classified as MIMD, SIMD and MISD[14]. In[15] a new parallel system is described, that combines both SIMD and MIMD architectures. This hybrid system consists of cluster of workstations and SIMD systems working concurrently to produce an optimal parallel computation. Massively parallel SIMD machines such as Distributed Array Processor (DAP), Connection Machine(CM-2) offer a high degree of parallelism to tackle computation intensive problems. Image processing has benefited from this architecture. Other computational tasks that produce voluminous amounts of data are better understood with the aid of scientific visualization techniques. It is an important tool to use a steering mechanism to control the course of a computation or simulation that needs some simple form of dynamic user interaction.

A parallel VLSI architecture for fast line drawing has been presented in[16]. The architecture implements a non-incremental line drawing algorithm, which allows writing simultaneously in a memory array all the pixels that approximate the straight segments. The bottom-up process for a simplified architecture design that reduces the circuitry redundancies in order to minimize the area, has been explained. This memory architecture also provides read/write random accesses and raster outputs that permit the memory architecture to display the data serially. A 256x256 eight- bit pixel processor array has been designed using 0.35mm standard cells. Test and simulation results have demonstrated that a rate of 50M segments per second can be achieved, independently of their length and orientation. An attempt to provide an improvement in line drawing speed, has been proposed. This is done by accelerating the process of obtaining the pixels to be drawn and to shorten the display memory write time[17].

A parallel algorithm is proposed which does not use the classical incremental techniques that obtain the present pixel coordinates from the previous ones. In this case, the computation to determine if a certain pixel belongs to a segment consists of a single comparison at every pixel between previous common computed values for each row and column of the screen. The maximum efficiency is obtained when placing one processor by pixel in which case the pixels that approximate the straight segment are obtained simultaneously.

The obtained parallel algorithm allows an implementation of a parallel architecture consisting in a matrix (M×N) of very simple pixel processors and N+M column and row calculation units. This architecture allows writing simultaneously in just one clock cycle all the pixels that approximate a given straight segment. Thus, the writing speed is only limited by propagation time of the technology used in the implementation.

Applying sequential line-drawing algorithms on parallel machines can lead to very inefficient codes. Ellsworth[18] describes an algorithm suited for interactive polygon rendering, where the model's image on screen has frame-to-frame coherence. He uses an optimized MIMD with all-to-all communications, where all processors communicate with all other processors[19]. suggests an improved parallel circle algorithm that uses equal x-step divisions. In[20] an incremental parallel line- and circle-drawing algorithm based on Bresenham's algorithm[21] has been presented. The algorithm uses only integer values and avoids any multiplication.

Some parallel machines possess the ability to support virtual processors. When the number of physical processors is not enough to cover the amount of parallelism in the problem, virtual processors give the illusion of a larger machine. This mechanism greatly reduces the amount of programming and increases the flexibility of the code for problems of different sizes. The ratio of virtual processors to physical processors is known as the VP ratio. When the VP ratio exceeds unity, the local memory in each physical processor is divided among the virtual processors assigned to that physical processor. Executing proceeds in parallel, but is time sliced among the virtual processors. Thus, when the VP ratio is n, the program will generally take n times as long to execute[14].

In section 2 we discuss briefly the main approaches used for designing reusable parallel software. Section 3 outlines some parallel computer graphics algorithms. Included here are our previously obtained results[20].

.Section 4 is concerned with the construction of reusable parallel graphics software. We give a template example for SIMD-MIMD hybrid systems with 4 PCs.

REUSABLE PARALLEL SOFTWARE

Without the ability to reuse existing algorithms and software, every programming project must start from scratch. Effective reuse technologies allow algorithms and techniques to be encapsulated in a reusable fashion as design patterns, functions, libraries, components, or objects.

Parallel software developers should identify classes of problems suitable for parallel implementation and develop efficient algorithms for each of these areas. With such a wide variety of computer systems and architectures in use or proposed, the challenge for people designing algorithms is to develop algorithms and ultimately software, that are both efficient and portable. To address this challenge, there are three, not mutually exclusive, approaches. The first approach is to express the algorithms in terms of modules at a high level of granularity. When moving software from one architecture to another, the basic algorithms are the same, but the modules are changed to suit the new architectures. A second approach is to create a model of computation representing the computing environment. Software is written for this model and then transformed to suit a particular realization of an architecture that fits the model. The general categories of MIMD and SIMD are of course too crude, but additional details could be specified. For example, an MIMD model might be characterized by the number of processors, communication vehicle, access to shared memory and synchronization primitives. Software written for such a model can be transformed to software for a specific machine by a macro processor or a specially designed preprocessor. As a third approach, the software can be written in high-level language constructs, such as array-processing statements. Again, a preprocessor can be written to generate the object code suitable for a particular architecture.

Of the three approaches, it is preferable to express the algorithms in terms of modules with a high level of granularity. In particular, it would seem applicable to certain basic software library subroutines that are expected to shoulder the bulk of the work in a wide variety of calculations. Where successful, the effect of this will enhance both the maintenance and use of the software. Software maintenance would be enhanced because more of the basic mathematical structure would be retained within the formulation of the algorithm. Software users may move existing codes to new environments and experience a reasonable level of efficiency with minimal effort. For this approach to succeed, a level of granularity must be identified that will permit efficient implementations across a wide variety of architectural settings. Individual modules can then be dealt with separately, retargeting them for efficiency on quite different architectures. This conceal the peculiarities of a machine from the user and will allow him to concentrate his effort on his application. This aims to exploit the key features of the architecture. This is where the approach based upon a model of computation can be useful. Research efforts should focus on generic multiprocessor algorithms that can be easily transported across various implementations of these designs. If a code has been written in terms of high-level synchronization and data-management primitives, which are expected to be supported by every member of the model of computation, then these primitives only need to be customized to a particular realization. A very high level of transportability may be achieved through automating the transformation of these primitives. Software maintenance benefits from the isolation of synchronization and data-management peculiarities.

The concept of a design pattern such as 'divide and conquer' in sequential programming has emerged as an approach to cataloging and communicating basic programming techniques. A specification of this pattern might specify the problem-independent structure and note where problem-specific logic must be supplied. This specification does not provide any executable code but provides a basic structure that can guide a programmer in developing an implementation.

Software developers use Component-Base Reuse (CBR). Fig. 1 shows a high-level model of a component interaction with its environment[3].

In parallel processing, the aim is to construct an algorithm template which is portable, scalable and adaptable to application requirements, yet retains the properties which make the algorithm desirable in the first place.

In practice there are a small number of basic parallel algorithm techniques. For example, the manager/worker structure is often appropriate when a large number of independent tasks need to be executed. A single manager process generates tasks and allocates those tasks to a number of worker processes each worker repeatedly requests tasks from the manager and executes the problem-specific code required to perform those tasks (returning results to the manager) until the manager signals that no tasks remain. Variants of this basic pattern may create a hierarchy of managers, in order to avoid a bottleneck at the central manager and/or allow for constant input data to be cached within workers, hence avoiding redundant communication.

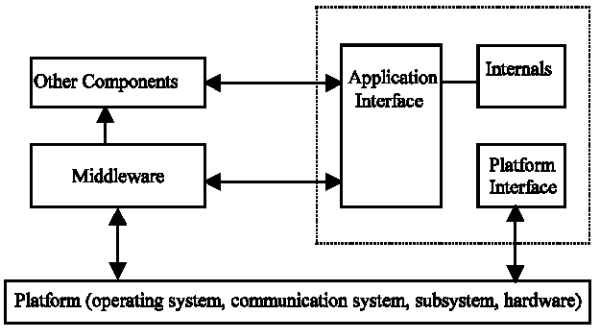

Fig. 1: The component model

Other common patterns include the butterfly, used, for example, to perform parallel summations in time proportional to the log of the number of processors and domain decomposition, which is of course fundamental to the data-parallel programming model. A template can be thought of as a reusable algorithm that includes the algorithm itself, along with information about how it is to be used, where specific computational specialization can occur and how the algorithm can be tuned. A template may also include some sample implementations in different languages and pointers to background information[2].

MPI's communicators mechanism allows the programmer to encapsulate communications that are internal to a function, hence avoiding race conditions that might occur if communications intended for one function are intercepted by another. This mechanism makes it easier to construct components so that interactions occur only via well-defined interfaces.

Each component has a name, model parameters, that may have the form:

type:: cname

The body of the component consists of an interface and a list of statements that define its function. Improved software engineering techniques allow data distribution issues to be separated from other aspects of function logic. What are sometimes called data-structure-neutral libraries allow an application to invoke an operation on a parallel data structure without regard to how the data structure is distributed; the distribution should impact performance but not correctness.

The increasing availability of advanced-architecture computers is having a significant effect on algorithm research and software development

Furthermore, many problems cannot be solved with existing "blackbox" software packages in a reasonable time or space. This means that more special-purpose methods must be used for the problem. There are a large number of tuning options available and for many problems it is a challenge to get any acceptable answer at all or to have confidence in what is computed. The expertise regarding which options are likely to work in a specific application area is distributed among many experts.

Thus, there is a need for tools to help users pick the best algorithm and implementation for their problems, as well as expert advice on how to tune them. This leads to algorithm templates, with a decision tree to help choose among them. The decision tree uses information about the structure of the problem, the kind of solution that is desired and the kind of computer available to identify one or more suitable algorithm templates.

The developers of algorithms have the need to get involved in the software development process. Robustness, ease of use and portability are standard fare in any discussion of algorithm design and implementation. As new architectures evolve, they will enclose concurrent processing, shared memory, pipelining, in order to increase performance. The portability issue is important because of the variety of architectural designs become reality. In fact, it is very tempting to assume that an unavoidable byproduct of portability must be an unacceptable degradation in the level of efficiency on a given variety of machine architecture.

Architectures of future machines promise to offer a profusion of computing environments. The existing forerunners have already given many software developers cause to reexamine the underlying algorithms for efficiency's sake. However, it seems to be an unnecessary waste of effort to recast these algorithms with only one computer in mind, regardless of how fast that one may be. The efficiency of an algorithm should not be discussed in terms of its realization as a computer program. Even within a single architecture class, the features of one system may improve the performance of a given program, while features of another system may have just the opposite effect.

PARALLEL GRAPHICS SOFTWARE

Assume that the line segment goes between the integer points (x_1,y_1) and (x_2,y_2) and that the slope satisfies $0 \le m \le 1$. The line can be represented as

$$y = mx + b, \quad (1)$$

where

$$m = \frac{y_2 - y_1}{x_2 - x_1}, \quad b = y_1 - mx_1$$

For any given x-interval Δx we can compute the corresponding y-interval $\Delta y = m.\ \Delta x$.

Eq. 1 includes a floating-point multiplication.

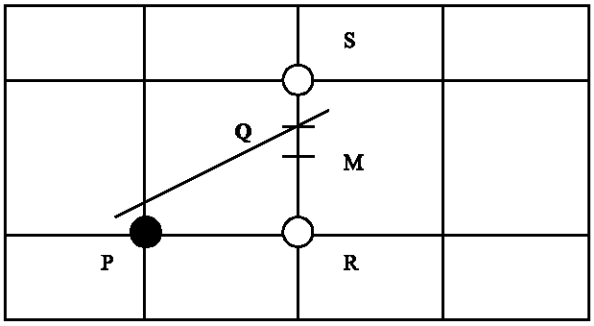

Fig. 2: The pixel grid for the Bresenham's algorithm

To avoid this operation we can use the Digital Differential Analyzer algorithm (DDA). Beginning with the left end of the line segment and considering $\Delta x = 1$, we can compute the successive y values as

$$y_{k+1} = y_k + m \qquad (2)$$

A pseudocode for the DDA algorithm is:

```
Algorithm 3.1 DDA Line Drawing
#define round(a) ((int)(a+0.5))
for (int ix = x1 ; ix <= x2, ix++)
{
  y = y + m;
  write_pixel (x, round(y), line_color);
}
```

The DDA algorithm eliminates the floating-point multiplication in (1) and therefore it is faster in calculating the pixel positions. The DDA algorithm requires a floating-point addition for each pixel. The following Bresenham's algorithm avoids all floating-point calculations and has become the standard algorithm used in hardware and software rasterizers[21-23]. To illustrate Bresenham's algorithm we adopt an approach given in[24]. We start from the left endpoint (x_0,y_0) and step to each successive pixel. At (x_k,y_k) we want to choose the next pixel. The condition $0 \le m \le 1$ allows only the two choices

$$(x_k + 1, y_k) \text{ and } (x_k + 1, y_k + 1)$$

In Fig. 2 let $P = (x_p,y_p)$ be the previously selected pixel, R and S are the two pixels from which to choose at the next step. Let Q be the intersection point of the line to be drawn with the line $x = x_p + 1$ and M be the midpoint between R and S. If M lies above the line, pixel R is closer to this line, otherwise point S is closer. The error of this test is ≤ 0.5. Now we need to calculate on which side the midpoint M lies. We write the equation of the line in the forms

Table 1: Increments for some slopes

x	Increment for s = 0.25	Increment for s = 0.75	Increment for s = 1.00
1	1	1	1
2	0	1	1
3	0	1	1
4	0	0	1
5	1	1	1
6	0	1	1
7	0	1	1
8	0	0	1
9	1	1	1
10	0	1	1
11	0	1	1
12	0	0	1

$$F(x,y) = ax + by + c = 0$$

and

$$y = (dy/dx)x + B, \text{ where}$$

B is the y increment, $dy = y_1 - y_0$ and $dx = x_1 - x_0$.
Comparing the two forms we get

$$a = dy,\ b = -dx,\ c = B.dx.$$

Let

$$d = F(x_p + 1, y_p + \tfrac{1}{2}) = a(x_p + 1) + b(y_p + \tfrac{1}{2}) + c.$$

If $d > 0$ (i.e. M lies below the line) then we choose pixel S, otherwise we choose pixel R.

For the next grid line, if R is chosen, M is incremented by one step in the x direction. The new value of d is

$$d_1 = F(x_p + 2, y_p + \tfrac{1}{2}) = a(x_p + 2) + b(y_p + \tfrac{1}{2}) + c.$$

It follows that $\Delta_R = d_1 - d = a = dy$.
Similarly, if S is chosen, M is incremented by one step in both directions. The new value of d is

$$d_1 = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c.$$

It follows that

$$\Delta_S = d_1 - d = a + b = dy - dx.$$

For the start point (x_0, y_0) it is easy to show that

$$\Delta_{start} = a + b/2.$$

Table 1 shows the increment value for lines with slopes s = 0.25, s = 0.75 and s = 1.00[20]

REUSABLE PARALLEL GRAPHICS SOFTWARE

Following[2] we give an example for a parallel graphics template that will include:

- A high-level description of an algorithm as in section3.
- A description of when it is effective, including conditions on the input and estimates of the time, space, or other resources required. Here, it must be clear, that the scanning goes from left to right. The number of input tables is proportional to the screen resolution.

Table 2: Pixel-table for the line with slope 0.25

x	y
60	60
61	61
62	61
63	61
64	61
65	62
66	62
67	62
68	62
.	.
.	.
.	.
200	95

As an extension to our algorithm,[20] we can generate a pixel table for the required line. As an example, the pixel table for the line with slope 0.25 between the points (60,60) and (200, 95) is:

We can assign to each processor a column of pixels. The processor the searches the pixel table for a pixel that lies on this column and sets it.

- A description of when it is effective, including conditions on the input and estimates of the time, space, or other resources required. Here, it must be clear, that the scanning goes from left to right. The number of input tables is proportional to the screen resolution.
- Pointers to complete or partial implementations, perhaps in several languages or for several architectures (such as different parallel architectures)
- A way to assess the accuracy
- Examples, on a common set of examples, illustrating both easy cases and difficult cases
- Troubleshooting advice
- Pointers to texts or journal articles for further information.

The template skeleton may have the form:

```
#The User Interface
# Fill in the   attributes.
# Example
# picked up by default, although the user can specify them if he wants to. For
int NumberOfPCc = 4;
int SystolaNumberOfProcessors = 1024;
# The Communication  mode
#Example
void broadcast_row (int n; selector s)
{
  # Code to set CS,c; s
}
# or alternatively
void  ringshift_row ( int n; int k; selector s)
{
 n = 1;
 while (n <= k)
  {
   #  Code to set CE, C, s
   #  Code to set CW, C, s
   n++;
  }
}
# Internals
# to set a pixel
void Par_Set_Pixel ( array *A; int startx; int starty)
{
   int x = startx;
   int y = starty;
   for int i = 1; i <= 4; i++)
     {
        #  Number of processors = 1024
        for ( int j =1;  j <= 1024; j++)
           {
             set_pixel (x, y + A[i + j]);
             x++;
           }
     }
}
```

CONCLUSIONS

Representing generic parallel templates as re-usable components for parallel algorithms aims at providing application-independent library of templates that facilitates extendibility of software systems and uses generic interfaces. We have presented a set of templates which can provide high granularity parallel applications on a MIMD-SIMD hybrid environment. The goal is to support templates for a cluster where some of the nodes in the cluster can have different number of PCs To achieve this, templates for shared-memory based algorithms and the necessary refinements in the model in these templates, need to be investigated. In the field of parallel line-drawing algorithms, we have extended our previous results to generate a pixel-table for each line. The template and protocol implementations and the user's interface of the development system is being refined to improve performance and the usability of the system.

REFERENCES

1. Selby, R.W., 2005. Enabling reuse-based software development of large-scale systems. IEEE Transactions on Software Engineering, 31: 495-510.

2. Dongarra, I. *et al.,* 2003. Sourcebook of parallel computing, Elsevier Science, USA.
3. Mili, H., A. Mili, S. Yacoub and E. Addy, 2002. Reuse-based software engineering. John Wiley and Sons, INC. New York, 2002.
4. Erwig, M. and Z. Fu, 2005. Software Reuse for Scientific Computing Through Program Generation. ACM Transactions on Software Engineering and Methodology, pp: 14-2.
5. Eilinghoff, C. and U. Kastens, 1998. Reuse methods for construction of parallel software. Lecture notes in computer science. Proceedings of the 25th conference on current trends in theory and practice of informatics: Theory and Practice of Informatics. pp: 56 - 67.
6. Geist, A. and V. Sunderam, 1992. Network-based concurrent computing on the PVM system. concurrency: Practice and Experience, 4: 293-311.
7. Gropp, W., E. Lusk and A. Skjellum, 1994. Using MPI: Portable parallel programming with the Message-Passing Interface. The MIT Press, Cambridge, Massachusetts.
8. Bacon, D.F., S.L. Graham and O.J. Sharp, 1993. Compiler transformation for high-performance computing. Technical report UCB/CSD-93-781, University of California, Berkeley.
9. Singh, A., J. Schaefer and D. Szafron, 1998. Experience with parallel programming using code templates. Concurrency: Practice and Experience.
10. Gamma, E., R. Helm, R. Johnson and J. Vlissides, 1994. Design patterns, elements of reusable object-oriented software, Addison-Wesley.
11. Siu, S. and A. Singh, 1997. Design patterns for parallel computing using a network of processors. In Sixth IEEE International Symposium on High Performance Distributed Computing, pp: 293-304.
12. Hansen, P.B., 1995. Parallel programming paradigms, Prentice Hall.
13. Chandy, K.M., 1994. Concurrent Program archetypes. In International Parallel Processing Symposium.
14. El-Rewini, H. and M. Abd-El-Bar, 2005. Advanced computer architecture and parallel processing, John Wiley and Sons.
15. Sim, L.C., H. Schroder and G. Leedham, 2003. MIMD-SIMD hybrid system- towards a new low cost parallel system. Parallel Computing, 29: 21-36.
16. Martí, P.M., A.B.M. Velasco, 2000. Memory architecture for parallel line drawing based on non incremental algorithm proc. EuroMicro'pp: 12-66
17. Mares, A.B. and J. Martinez, 2004. Aranda parallel co-processor for ultra-fast line drawing PARA'04 State-of-the-Art, in Scientific Computing.
18. Ellsworth, D., 1994. A new Algorithm for interactive graphics on multiprocessors. IEEE Computer Graphics and Applications, pp: 33-40.
19. Huang, J and E. Banissi, 1997. An Improved parallel circle-drawing algorithm IEEE Computer Graphics and Applications, pp: 40-41.
20. Meligy, M. Aly and H.A. Nassar, 1997. A parallel algorithm for computer graphics on systems with shared memory ann. Conf. ISSR, 33:141–157.
21. Angel, E., 2003. Interactive computer graphics, Addison Wesley, New York.
22. Alex, T., 1990. Pang line-drawing algorithm for parallel machines. IEEE Computer Graphics and Applications, pp: 54-59.
23. Schimmler, M., H.W. Lang, 1996. The instruction systolic array in image Processing applications. In: Proceedings Europto 96, SPIE 2748, pp: 136-144.
24. Foley, J.D. *et al*., 1997. Computer graphics, Addison-Wesley.