A Review of Parallelization Tools and Introduction to Easypar

Sudhakar Sah
Symbiosis Institute of Research & Innovation (SIRI)
Pune, India

Vinay G. Vaidya
Symbiosis Institute of Research & Innovation (SIRI)
Pune, India

ABSTRACT

Multicore processors have paved the way to increase the performance of any application by the virtue of benefits of parallelization. However, exploiting parallelism from a program is not easy, as it requires parallel programming expertise. In addition, manual parallelization is a cumbersome, time consuming and inefficient process. A number of tools proposed in the past ease the effort of parallel programming. This paper presents a classification of such parallelization tools. The classification is based on different eras of tool development, role playedby these tools in various parallelization stages, and features provided by parallel program assistance tools. Classification of tools concludes with a discussion on requirements of futuristic parallelization tools. Finally, this paper proposesour on-going work about the development of a parallel program assistance tool called EasyPar, which is a parallel program assistance tool.

General Terms

Parallel Processing, Parallelizing Compilers, Multicore

Keywords

Interactive Parallelization, Parallel Program Assist, Automatic Parallelization, Parallel Programming Tools, Multicore.

1. INTRODUCTION

Parallel programming is not a new concept. In fact, it started in early 1970s and handful of techniques proposed during that time are still used [1], [2],. Now, the question does arise in one's mind, why is the research focus back on parallel programming aftermany decades?. Surprisingly, now the amount of research is much more as compared to its early 1970s research efforts. The answer to this question is that today general programmers need parallel programming, as opposed to scientific researchers in the early days.Need of parallel architecture such as multicore arose due to the limitations in increasing the clock speed of processorsas per the famous Moore's law [3]. The underlying reason of this limitation is that the heat dissipation increases proportionally or more by increasing clock speed for constant chip area. Therefore, the chip manufacturers came up with the multicore processor having more than one core fabricated on the same silicon chip. The introduction of multicore theoretically enabled the speed of a processor by the multiple of number of available cores. However, invention of multicore processor further complicated the scenario [4] as most of the legacy application iswritten in sequential manner and hence are incapable to utilizing the true power of multicore. This limitation demands the use of parallel programming. Unfortunately, most of the programmers are naïve or unaware of parallel programming concepts [5]. Parallel programming training is not feasible from cost and time perspective. Therefore, there is a growing need of tools that can assist in parallel programming. A number of tools and techniques are available in literature which targets to ease the parallel programming. This paper provides a brief review of existing parallel programming tools. This paper contributes to the classification of existing tools based on three aspects. First classification is based on whether the tool was developed before or after the invention of multicore. We will make it clear during the review of these tools that there was a fundamental difference in the way researchers thought, before, during,and after the multicore era. Program parallelization is a stepwise process, as explained in section 3. The second classification places these toolsaccording to theircontribution during the process of parallelization. Garcia et. al. [6], [7], [8], have presented similar but limited review and classification of such tools. However, this review includes most of the relevant tools proposed in recent times as well as significant older tools. In addition, these tools are sub classified based on the parallelization technique; i.e. it supports loop parallelization, task parallelization or both. The third and final classification is based on whether the tool provides parallel programming assistance or not. Section 3 discusses features and demerits of some of the important tools, and based on the discussion, we presentthe viewpoint on the requirement of future parallelization tools. This paper concludes with explanation of the ongoing work on a tool called EasyPar[9], [10]. EasyPar is a parallel programming assistance tool that helpsdevelopers at the time of program development. We discuss the challenges in the development of tool such as EasyPar. Finally, methodsto overcome these challenges is proposed.

2. PARALLELIZATION TECHNIQUES

Parallelization can be achieved in many different ways as shown in Table 1. This section, presents an overview of these techniques in this section. Table 1 also compares these tools qualitatively based on the time required for parallelization, learning efforts, and efficiency of generated parallel code.

2.1 Automatic Parallelization

Automatic parallelization,[25], [27], [30], [31], [32], [33], [39], [43], [45], [51], [89] , [98] as the name suggests adaptstechniques that accept a serial source code and returns a fully parallelized source code. An intelligent analysis engine [7], [8] [58], [82], running in the background does the trick of parallelization. The intelligent engine includes static analysis of code fordata dependency [59] check among different code segments. The dependencecheck depends upon the code semantics and it creates the groupof code segments withpossibility of concurrent execution. Although, the technique looks attractive and fascinating, it has inherent limitations associated with coding style. First reason is that such tools check for program semantics and not the core logic of the program (which itself is an open research problem – "Optimistic parallelism requires abstractions" by Kulkarni [60]). Therefore, parallelization achieved by the use of this technique is limited and it misses many possible parallelization opportunities. Another limitation of automatic parallelization tools is that it is limited to target language, features, platform etc. Nevertheless, automatic parallelization is very important because it requires less time to convert the sequential code to parallel code. In addition, it removes the burden of parallelization fromprogrammer. The degree of parallelization achieved using this technique solely depends upon the code and the intelligence of the analysis engine. Ryder [22] et al has provided a brief review of the work done in compile-time program analysis.

Table 1:Classification of parallelization techniques

Technique	Time	Learning	Parallelization
Automatic Parallelization	Low	Low	Code dependent
Semi-automatic Parallelization	Moderate	Moderate	Code Dependent
Assistance tools	High	High	High
Hardware Support	High	High	Very High
Parallel Languages	High	High	Very High

2.2 Semiautomatic Parallelization

Semiautomatic parallelization techniques[7], [8],[44], [62], [73], [74], [75],[77],[78], [82],[86], [88], [91] do not provide end-to-end code parallelization option. Such techniques do not believe in using the application asa black box for parallelization. It requiresvital information about the application/code from the programmer or user in order to take critical parallelization decisions such as, loop count, information about variable usage, branch prediction information and so on. Such information may not be available during static analysis of code (used for automatic parallelization). Parallelization decisionsthat rely on these inputs increase the opportunity of exploiting parallelization. This techniqueproduce the code with high degree of parallelism compared to the automatic method. However, it expects programmersto have limited knowledge of parallelization and mapping those concepts to the program logic. Such tools are very much useful when the program flow depends predominantlyupon the user inputs. [29]

2.3 Parallel Programming Languages

Most of the automatic and semi-automatic tools work for popular languages like C, C++, and FORTRAN etc. However, these languages are inherentlynot suitable to write concurrent programs. In the past, many programming languages have been developed specifically to develop parallel codes. One of the significant examples of such technique is functional programming. Functional language is conventionally different from other languages like C. The programming paradigm allows writing parallel programseasily. Haskell[34], Erlang [92], Cilk [93], Go [94] andScala [95] are few examples of parallel programming languages. Few languages like Jade [52] also provide the facility of machine independent parallel programming.However, all of these languages require a different approach to programming and it is hard for programmers who are used to thinking and writing sequential code to think parallel. Thus, these languages have not progressed to the extent they should have.

2.4 Hardware Support

Conventional memory uses lock based mechanism and processes can take exclusive lock of writable memory locations. This prevents other processesthat require reading from same memory locations to proceed further. This is true even when the other process reads index where first process writes. This is a huge bottleneck in concurrent programming as many process remains in waiting state unnecessarily. Transactional memory(TM) [35],[87],[96] provides an elegant solution to this problem. It works on the concept of transaction inherited from databases. Every process starts its transaction (independent piece of task) by using private copy of variables. After some time, all the processes check for conflict situation. Conflict is a situation where one variable or memory location has two different private values. In such situation, processes roll back (cancel) the operation and repeat it again. In case no conflict is detected, process commits (completes). An advantage of transaction memory is that any program can execute concurrently and TM will take care of concurrent execution. However, the performance of program penalizes in the presence of large number of conflicts. The research in TM is still in infancystage and development of hybrid TM (both software library for TM and hardware TM) is in progress that will possibly make usage of TMpossible.

2.5 Parallel programming APIs

Apart from techniques mentioned above, many programming APIs are available that support parallel program development. Out of these, Message Passing Interface (MPI) [1] and Open MP [2] are few of the most popular and older APIs. These APIs expect programmers to identify the parallel program segments and use the APIs for concurrent execution of program. MCAPI from Multicore association [36] is a suite of parallel programming APIs for multicore processors. CUDA (Compute Unified Device Architecture)[37] isa programming technique to harness massive parallel programming capabilities of NVIDIA GPGPU (General Purpose Graphical Processing Unit). CUDA provides specially designed APIs along with the hardware support that makes parallel programming easier and fruitful for data parallel programs. IMAPCAR [20] is also data level parallel architecture that uses C-like language to develop parallel programs.SWARM [28] is another tool with programming APIs for multicore. Intel has recently developed parallel programming APIs called threading building blocks (TBB) [96] that provides exclusive constructs to hide the multithreadingrelated burden from user. Nevertheless, most of these APIs push the burden of identification of concurrent code to the developer.

3. CLASSIFICATION OF PARALLELIZATION TOOLS

Parallel programming tools are continuously evolving and the evolution is highly influenced by the advancement in hardware. We present three different classifications of parallel programs in the review work. This sectionhighlights the basis of theproposed classifications.

3.1Classification based on parallelization stages

Parallelization process is a systematic process [6] (especially automatic parallelization) as shown in figure 1. First stage of the parallelization process is parallelization identification. The code is parsed and analyzed (static or dynamic dependency analysis [7], [8], [58], [82], [99]) to search for the code sections that can be executed concurrently. Apart from dependency checks, profiling of code is also done to identify hot spots. Hot spots are the sections of code where code spends most of its time. Kremlin [6]is one of the most important data dependence profiler developed in recent times. This stage is most challenging and complex because of the variation in code style, type of code, complexity of the algorithm (static analysis finds it difficult [18]) and lack of information available during code analysis (data values). This stage builds the foundation of further stages and next stages uses the data gathered during identification stage.

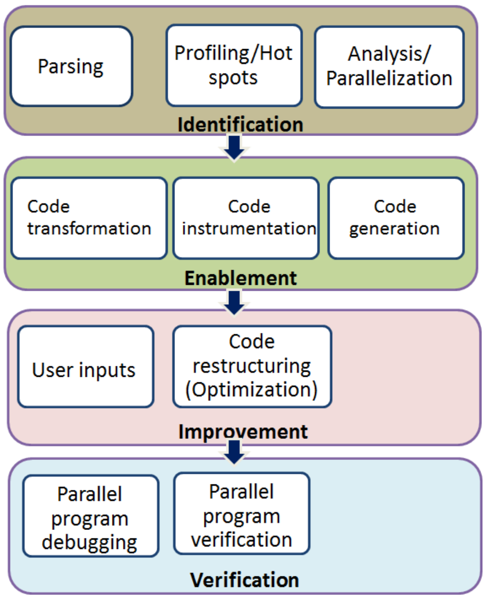


Fig 1: Stages of parallelization

Second stage of parallelization is parallelization enablement. Concurrent code identified during the first stage needs to be scheduled to execute on different cores/ processors.Enabler schedules the concurrent code to different cores. It is achieved by using thread mechanism provided by OS or using of the shelf APIs like MPI [1], Open MP[2], TBB [96] etc. These APIs provide simple interfaces and pragmas to take off the burden of writing multithreaded code. Parallel code developed until this point may not be optimal and there exists lot of scope for improvement. In addition, parallelization is achieved using first two stages without any code transformation of the code. Code transformation technique removes the dependency among code segmentsand increases the possibility of concurrent execution. Prospector [91], Kremlin[6], Intel Parallel advisor [72], Cilkview[85]ParaAssist [38]and Alchemist [7]employ code transformation techniques for improving parallelization possibilities..

Parallel code generated using above mentioned techniquesneedverification. The verification stagetests whether behavior of parallel version of the code is exactly same asthe serial version or not. This step may involve the debugging of the parallel code. Tallet et al [83] has proposed a method to measure the performance of multithreaded program. Quartz [84] is another tool for performance tuning of parallel programs. ThreadSanitizer [69], MS concurrency visualizer [70], Chess [67], Racetrack [66], Ctrigger [65], Perver, Prism [73], EasyPar[9],[10]and Kismet [79] helps in verification and debugging of parallel codes. Figure 4 provides the list of tools based on parallelization stages.

3.2Classification based on the Era

We divide the development of parallelization tools into two eras. We define the tools developed in these two eras as first-generation (FGT) and second-generation tools (SGT).

3.2.1 First Generation Tools (FGT)

Development of First generation of tools happened before the invention of multicore. Figure 2 shows a simple example of loop level parallelization.Every element of array B is added by 3 and the result is stored in an array A. Array C is populated by a constant value returned by module(). Both of these operations can be performed independent of iteration. Such type of parallelization is termed as loop level parallelization.

Distributed systems weremainstreamparallel hardware during the development of FGTs. Since, there was a lot communication overhead due to data transfer between distributed machines;vectorization was popular concept at that time. Vectorization is the technique, whichis used for loop parallelization. Therefore, most of the parallelization tools developed during FGT exploited parallelism offered by loops. Another reason of focus of loops to exploit parallelism is that parallel processing concept was used predominantly for scientific applications and simulations. Most of these applications needed repetitive computation on same data or same function repeated for multiple times.Very few task parallelization tools were developed during the first generation.SUIF [25], [43], RawCC, Polaris [88], CAPO [101], Para Assist[38],OpenMP [2], MPI [1]are some first generation tools and most of them focus on loop level parallelization. However, OpenMP and MPI APIs that enables parallel code generation can also be used for task level parallelization.

```
For(i=0;i<2000;i++) {
A[i]=B[i]+3; //iterations are independent
C[i]=module(3);
}
```

Fig 2: Loop level parallelization example

```
int main {
int i, j, k;
i=100;
j=5;
foo(&i);  // i is modified in foo()
check(&i);// j is modified in check()
k=finalize(i,j); //dependent on foo() &
check()
}
```

Fig 3: Example of task level parallelization

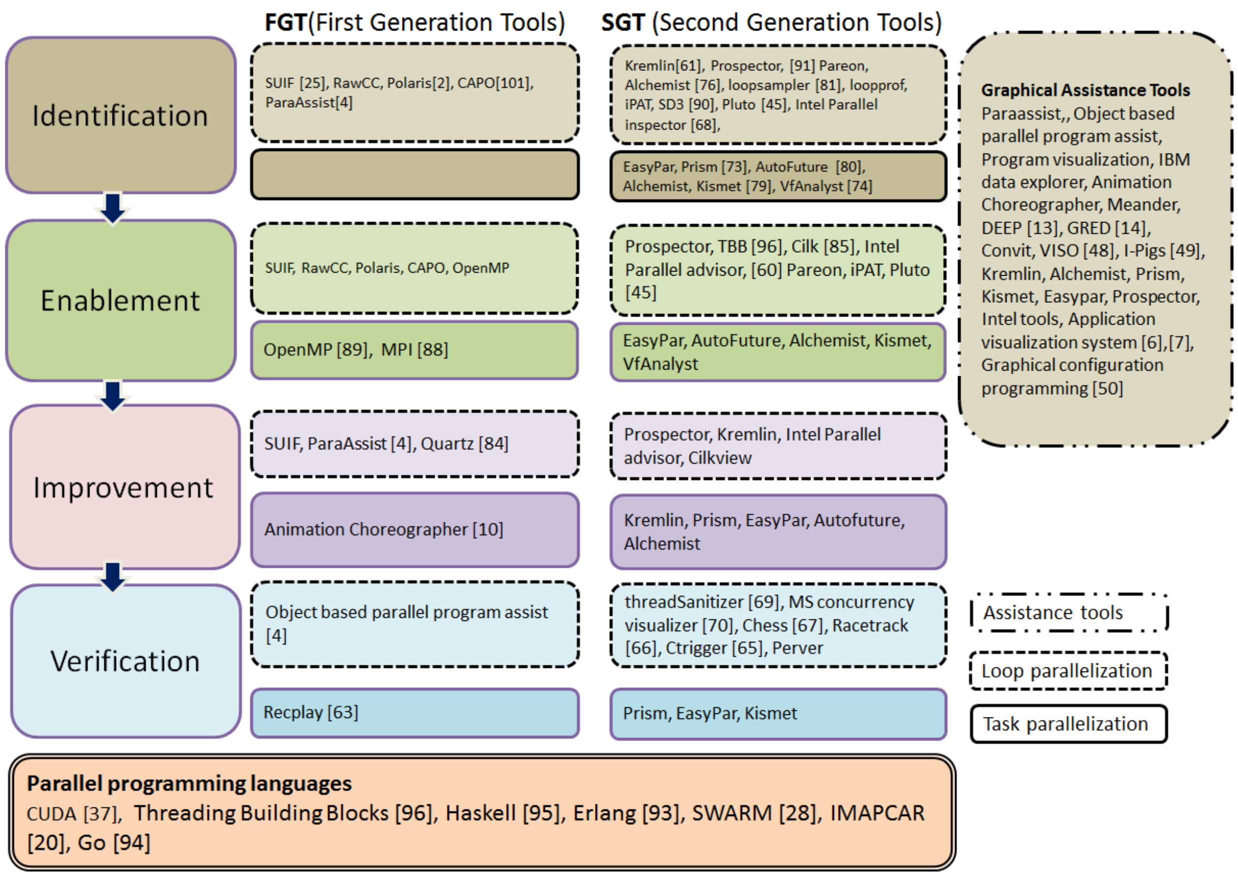

FGT(First Generation Tools)
SGT (Second Generation Tools)
Identification
SUIF [25], RawCC, Polaris[2], CAPO[101], ParaAssist[4]
Kremlin[61], Prospector, [91] Pareon, Alchemist [76], loopsampler [81], loopprof, iPAT, SD3 [90], Pluto [45], Intel Parallel inspector [68],
EasyPar, Prism [73], AutoFuture [80], Alchemist, Kismet [79], VfAnalyst [74]
Graphical Assistance Tools
Paraassist,, Object based parallel program assist, Program visualization, IBM data explorer, Animation Choreographer, Meander, DEEP [13], GRED [14], Convit, VISO [48], I-Pigs [49], Kremlin, Alchemist, Prism, Kismet, Easypar, Prospector, Intel tools, Application visualization system [6],[7], Graphical configuration programming [50]
Enablement
SUIF, RawCC, Polaris, CAPO, OpenMP
Prospector, TBB [96], Cilk [85], Intel Parallel advisor, [60] Pareon, iPAT, Pluto [45]
OpenMP [89], MPI [88]
EasyPar, AutoFuture, Alchemist, Kismet, VfAnalyst
Improvement
SUIF, ParaAssist [4], Quartz [84]
Prospector, Kremlin, Intel Parallel advisor, Cilkview
Animation Choreographer [10]
Kremlin, Prism, EasyPar, Autofuture, Alchemist
Verification
Object based parallel program assist [4]
threadSanitizer [69], MS concurrency visualizer [70], Chess [67], Racetrack [66], Ctrigger [65], Perver
Recplay [63]
Prism, EasyPar, Kismet
Assistance tools
Loop parallelization
Task parallelization
Parallel programming languages
CUDA [37], Threading Building Blocks [96], Haskell [95], Erlang [93], SWARM [28], IMAPCAR [20], Go [94]

Fig 4: Classification of Parallelization Tools

3.2.1 Second Generation Tools (SGT)

Parallellization tools development scenario completely changedafter the invention of multi core. Multicore is shared memory architecture. At present, most machines with 4 to 8 cores are available commonly, though for scientist applications uses multicores with much more number of cores. The shared memory architecture comes with an advantage that the tasks can use common memory during execution and this lead to the popularity of task level parallelization. Therefore, programmers or researchers started developing techniques and tools to divide the program into concurrent tasks. These tasks may or may not be part of the loops. As shown in figure 3, function foo() and check() modifies values of variable i and j respectively. Therefore, they are candidates for concurrent execution. However, function finalize() updates values of both i and j, which makes it impossible to execute concurrently with foo() and check(). Concurrent execution of functions or tasks (group of statements) is knownas task parallelization.Kulkarni et. al. talks about parallelization in irregular application [86] which means both task and loop parallelization, specifically application which is not easily parallelizable. Some of the examples of such tools are AutoFuture [80], Prism [73], Kremlin[6], EasyPar[7], Alchemist[7], VfAnalyst [74] etc. Some of these tools are explained later in this section.

3.3Graphical Assistance Tools

According to the parallel programming tool categorization given in section 1, one type falling under these categories provides the information about program flow and/or application flow graphically. This section explains techniques that reduce the programmer's burden by providing vital program information graphically that can help to develop high quality parallel code. Graph based interactive program analysis tools arefurther dividedinto two categories based on the information that it generates. [88]

• Static program information – Such tools display the information about data structure, flow of data, data dependency etc. This information helps developers to design their parallel program better.

• Algorithm animation – Program information alone is not sufficient to develop efficient parallel programs. Programmers would rather appreciate tools that can provide some glimpse of application graphically. Kulkarni et. al [60] believes that optimistic parallelization requires abstractions, means knowledge about the algorithm and not only the static information.

A complete parallel program assistance tool requires above two qualities to cater to the future multicore programming requirements.

ParaAssist, Program visualization, IBM data explorer, Animation Choreographer[40], Meander, DEEP [13], GRED [14], Convit [46], VISO [48], I-Pigs [49] are first generation graphical assistance tools and Kremlin, Alchemist, Prism, Kismet, Easypar, Prospector[91]and Intel visualization tools are some example of second generation tools providing program visualization.

3.4 Summary of Classification

Figure 4 shows detailed classification of the parallelization tools based on the three aspects presented in earlier sub section. Tools listed in boxes with dotted and solid boundary work onloop and task parallelization techniques respectively. Tools supporting parallelization for a particular stage out of

the four mentioned earlier arelistedin boxes on right side at the same level (loop and task level parallelization tools separately). It is clear that very few tools were developed in the first generation that supports the verification of generated parallel code. Parallel program verification was done using manual techniques except that in few tools like object based parallel program assist [38]. This tool was specifically applicable for object-oriented programs and it informs about the side effect due to executionof concurrent program in presence of data dependency.
Graphical assistance toolsusually work on semi-automatic parallelization techniques. Such tools have two inherent advantages. First, it gives complete insight about the program, which helps programmer to write optimized parallel code;This is not possible in case of automatic parallelization. Second advantage of using such tools isthat their visual output and online assistance, proves to a parallel programming trainer. Graphical assistance tools gradually takes programmer to a level where he or she can think of developing parallel code from abstract level information about application [60].

Another classification not mentioned so far is the parallel programming languages like functional programming, CUDA, IMAPCAR etc. These languages are shown in the box having double line boundary. These languages work in fundamentally different way as compared to other sequential languages. These languages allow us to write parallel program from abstraction level instead of writing a program and then convert it to its parallel counterpart.

Next section will explain some of the important tool t mentioned in the tool classification.

4. PARALLEL PROCESSING TOOLS

This section discusses some of the existing parallelization tools and techniques. We have divided this section into two subsections. First subsection explains the tools and techniques developed recently (after the invention of multicore, i.e. second- generation tools). Second subsection explains some of the tools developed during earlier decades. These tools may not be in use as of today, however, it is important to discuss the tools in brief because they are the basis of development of second-generation tools. We have also mentioned the features and limitations of each tool.

4.1 Second Generation Tools (SGT)

4.1.1Alchemist

Alchemist [7]is a novel data-dependency analysis and profiling tool. It does not concentrate just on specific sections of a code (e.g. loops),rather, it explores possibilityof parallelism in all parts of the code. Alchemist does not rely on any specialized hardware or software system support. Alchemist also provides vital information about the code that is required for decision related to parallelization. It also performs code transformation to take care of WAW (write after write) and WAR (write after read) dependency. Therefore, it has the potential to help programmers in all four stages of parallelization and it is applicable to both task and loop parallelization.

4.1.2 DProf

DProfis a compiler driven approach for thread level speculative (TLS [64]) parallelization. The main contribution of DProfis a static model for TLS profitability that is used by the compiler to select independent tasks. Compiler is used to automatically perform the program dependent profiling. It proposes the concept of dependence clustering (region of iteration space having large independence window) and independence window (set of consecutive iterations that are independent of each other).

4.1.3 Prospector

Prospector [91] is a parallel program assistance tool especially developed for parallelization of loops. Prospector presentsa technique to reduce the loop level data-dependency by code instrumentation. In contrast to most of the other tools, which uses static dependence analysis, prospector uses the dynamic data dependence profiling. Dynamic data-dependence analysis technique has a limitation of scalability and it does not work for programs with large memory footprint. Prospector has used the compressive memory streams to handle this problem. Prospector performs sophisticated analysis apart from loop profiling to get accurate information about parallelization benefits.

4.1.4 Coarse grain parallelization

Coarse grain parallelization technique [75], [97] is yet another automatic parallelization technique. It has a distinction that it searches for code level parallelization. Code is divided into important segments that incudes loop, functions and others (code segments containing memory references). This technique uses the dynamic data dependence analysis.

4.1.5 LoopSampler

LoopSampler [81] Identifies potential parallelism in a program using loop centric profiling. Loop centric profiling provides hierarchical view of time spent in loops and loops nested within it. It involves two concepts, one based on instrumentation (used extensively in LoopProf) and other based on sampling approach. Sampling approach is novel contribution of LoopSampler. Sampling approach has significantly lower profiling overhead as compared to LoopProf.

4.1.6 iPAT/OMP

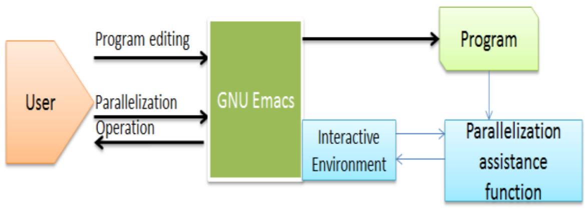


Fig 5: iPAT/OMP

IPAT Parallelizing assistance tool provides critical parallelization related information (loop centric). The system uses Omni Open MP compiler and its assistance libraries. The programming environment is split into two parts. First window is for program editing and second window displays the parallelization specific information to the user. User can also select the code segment for assistance and the tool displays the dependency information and assistsin resolving those dependencies.

4.1.6 Capo

CAPO [101] is an interactive parallelization and performance analysis tool developed by NASA Ames Research Center to insert OpenMP directive into FORTRAN codes. Paraver, [101]a performance analysis tool is developed by CEPBA-UPC to analyze the performance of a parallel program. Capo-Paraver is a computer aided parallel programming environment that interfaces CAPO with Paraver. This tool assists programmers in complex optimizations of parallel programs, which is very difficult manually. Capo has an in built dependency analysis engine for loops which has additional feature of storing the dependency information in a database and improving the dependency analysis by using answers to the questions asked to the user. Paraver consists of a tracing package and a graphical user interface for examining the traces. Paraver has the capability to analysis thread level, task level and hybrid parallel programs. Therefore, this environment helps user in both data dependency analysis and the performance analysis. It also takes input from user that increases the efficiency of parallel program. However, this tooltargets only the loop level parallelization and not the task level parallelization (analysis).

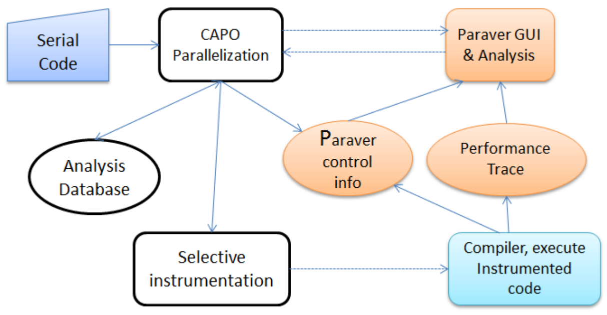


Fig6: CAPO parallelization [101]

4.1.7 Kremlin

Kremlin is one of the most significant works on automatic parallelization in recent times. Kremlin has proposed hierarchical critical path analysis (HPCA) for the first time, whichis being used for many of upcoming and existing profilers. Kremlin is able to exploit parallelism present in a program, which was not detected using existing critical path analysis (CPA). Kremlin also provides an OpenMP parallelism planner, which at times beats even the manual parallelization in terms of performance. It takes original source code along with other inputs and produces code regions that should be parallelized. Kremlin is applicable to all types of parallelization like task level, thread level, instruction level etc. Kremlin suffers from a major drawback (even other similar tools)of accuracy as it uses the dynamic run time information but Kremlin is able to overcome this limitation by multiple runs of the same program with different inputs.

4.1.8Kismet

Kismet [79] is an interesting tool for estimation of parallel program speed up. The speed up is computed based on the parallelism available in source code in presence of multiple constraints like, number of available cores, cache, shared memory size, synchronization overheads, parallelism types (Loop level, task level, instruction level) etc.Kismet applies the dynamic analysis using hierarchical critical path analysis (HCPA)[6]to determine parallel regions efficiently. HPCA is modified version of critical path analysis technique (CPA [47]), which is in use for quite a long time. It consists of two major components, one the self-parallelism profiler and second, the speed up predictor. Self-parallelism profiler instruments the code to obtain profiling information and to remove false dependencies in loops. Speed up predictor uses the profiling information from self-parallelism profiler and other hardware specific information to estimate the execution time of each code region. All the information from above sub blocks is used to geta consolidated speed up estimation. Though the tool talks about task and loop parallelization, it concentrates more on the loop parallelization offered by the program. Figure 6 shows sample output produced by Kismet. It is important to know that the maximum speed up is achieved for a four-core processor.Speed up is proportional to the number of cores up to four cores. However, speed up remains same when the program is executed on more number of cores and this follow the Amdahl’s law [23].

Cores	1	2	4	8	16	32	64
Speedup	1	2	3.8	3.8	3.8	3.8	3.8

Figure 6 : Typical output of Kismet [79]

4.1.8 Cilk++

Cilk++ [85], [92]concurrency platform helps programmer to use simple constructs in a program to parallelize a program. Cilk++ implements its own scheduler that takes care of parent child processes to be executed on different cores of multicore processor. However, Identification of parallelization in a code and synchronization point is responsibility of the programmer. Hence, this tool is parallelization enabler.

4.1.9 Holistic approach for automatic parallelization

Another parallelization approach uses profiling based dependency analysis [15] instead of using static code analysis method as shown in figure 7. Static analysis seems to be inefficient for parallelism identification and generation of parallel code. After identification of parallel segments, machine learning based mapping is used to generate OpenMP annotated parallel code. The approach works for loop level parallelization and not for the course grain task level parallel code segments.

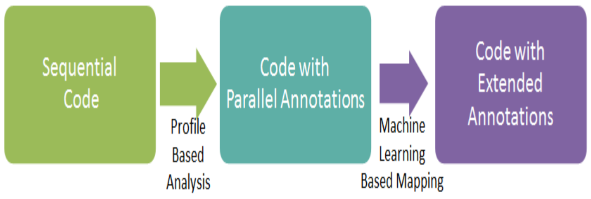

Sequential Code
Profile Based Analysis
Code with Parallel Annotations
Machine Learning Based Mapping
Code with Extended Annotations

Fig 7 : Holistic approach for automatic parallelization [15]

4.1.10 Polaris

Polaris [88] is another loop level parallelization tool that uses variety of loop dependency tests and loop transformation techniques to develop effective parallel code. It uses standard tests like equality test, GCD tests [99] for simple and linear cross iteration dependency. However, many real life code segments also contain nonlinear cross iteration dependency. Polaris solves this problem by using separate test called range test [88], which uses computer algebra and data range information to detect cross iteration dependency.

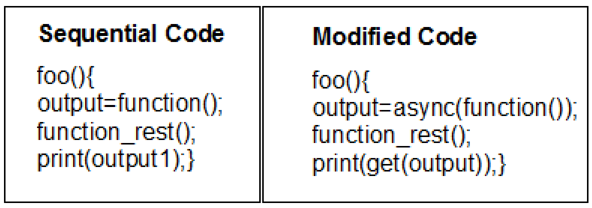
4.1.11 SD3

SD3 [8]presents a scalable approach to the data-dependence Profiling. Most of the data-dependence profiling techniques try to improve the accuracy of analysis with the help of user inputs or run time profiling. However, these techniques suffer from two major drawbacks; runtime overhead and the memory overhead. SD3 solves this problem by parallelizing the dependency analysis on multicore and achievesspeed up of above 9x over the earlier efforts. Similarly, the tool compresses the memory access exhibiting stride pattern and improves the memory consumption by 20x. Authors have contributed to the parallel algorithm design of data dependency profiling to overcome scalability problems.

4.1.12 Prism
Prism [73] is a commercially available parallelization tool (from Critical Blue) that supports the development of parallel programs on multiple fronts. First, it provides a profiler to detect the hot spots based on the time consumed and frequency of the code segment. After identifying the hot spots, prism can show the data dependencefor taking parallelization decisions quickly. Code segments/functionscan be selected for parallel execution and prism can show actual benefits of concurrent execution. Finally, the tool provides the facility to verify parallel code by providing information related to data races, dependency remaining in the parallel code segments etc. Prism supports in all stages of parallelization and it works for both taskas well as data parallelization. In addition, it serves as both automatic as well as assistance tool for developing parallel code.

4.1.13 AutoFuture
Autofuture [80] takes a completely different approach to concurrent execution of independent segment of codes. Two concurrent code segments executes with the help of synchronization points. However, insertion of synchronization points makes the code less readable. Autofutureproposesan elegant way to instead of using synchronization points. It executes two independent sections asynchronously and stores the result of the first one in a placeholder called 'future' [80]. This avoids the need of synchronization point by just the insertion of simple constructs. Figure 8 shows the concept used by Autofuture with the help of a simple example (use of 'async').

4.1.14 Vector Fabrics
Vector fabrics [74] is another commercially available tool (from Pareon) that gives insight information about the program which is crucial for writing highly optimized parallel code. The tool performs data- dependency analysis, convert it to parallel code for a particular architecture and then informs about the global performance data as well platform specific information like cache statistics. It also collects information about thread waiting overheads and provides suggestions onreducing these overheads. Last but not the least, the tool provides detailed guidance about code transformation to increase parallelization benefits and reduce other platform specific overheads. Vector fabrics isuseful for programmers with or without parallel programming expertise.



Sequential Code	**Modified Code**

foo(){
output=function();
function_rest();
print(output1);}
	foo(){
	output=async(function());
	function_rest();
	print(get(output));}

Fig 8 : Autofuture parallelization concept [80]

4.1.15 Pluto
Pluto [45] is a tool for source-to-source transformation of sequential code using available parallelism and locality. Pluto uses the polyhedral analysis, which is one of the most efficient loop parallelization, and transformation techniques developed in recent times. Most of the previous generation loop- parallelization tools (Before the advent of multicores) used standard loop dependency tests like GCD, Banerjee tests [99]. Earlier tests failed to identify parallelism available in loops due in presence of complex dependencies. Pluto emits the parallel code instrumented with OpenMP [2] constructs. Though polyhedral transformation was in use for some time, it lacked scalability and practicability. Pluto has improved the polyhedral transformation method [45] and solved earlier problems by developing a compiler that is capable of fully automatic parallelization.

4.1.16 Par4All
Par4All [11]is an automatic parallelizing and optimizing compiler for C and FORTRAN programs. It is based on PIPS (Parallelization Infrastructure for Parallel Systems) [100] source-to-source compiler framework. The 'p4a' is the basic script interface to produce parallel code from user sources. It takes C or FORTRAN source files and generates OpenMP [2] or CUDA [37] output to run on shared memory multicore processor or GPGPU respectively.

4.1.17 Cetus
Cetus[12] is a source-to-source transformation tool for programs written in C language. It also provides basic infrastructure to write automatic parallelization tools. Cetus currently implements parallelization techniques like are privatization, reduction variables recognition and r variable substitution. Cetus enables automatic parallelization by using data dependence analysis with the Banerjee-Wolfe inequalities [99], array, and scalar privatization.

4.1.18 S2P
The S2P [91] tool is commercially available(developed by KPIT Cummins) fully automatic parallelization tool that considers loops as well as tasks for parallelization. S2P is applicable for parallelization of legacy C program without any manual intervention. S2P performs the program analysis, identification of parallel segments and scheduling them to available cores of a multi core processor.

4.2 First Generation Tools (FGT)

4.2.1 Automatic and Interactive Parallelization
Kathryn et. al. [62] proposed an interactive technique for loop parallelization. This technique points out the inability of automatic parallelization technique due to the lack of information available at the time of static analysis. Interactive parallelization adds human insight, seeks important information, and receives it from the users to improve the parallelization results. The tool is called Parascope Editor (PED) and it provides option for user inputs to increase the chances of parallelization and to increase the accuracy of the analysis. To begin, the user selects potentially parallelizable loop and PED runs complete analysis and displays the dependencies in visual form as shown in figure. Based on the program understanding, user can mark some of those dependencies as false dependency. PED runs the analysis again based on user inputs and presents the parallelization report. Parallelizable loops without any ambiguity are marked as proven. The loop identified as non-parallelizable due to over conservative analysis is marked as pending. User analyzes the code and based on the program understanding mark the pending loops as accepted. Loops that are marked as pending are transformed according to the dependency inputs.

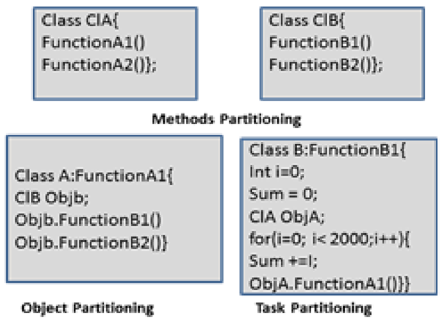
Users just need to put assertions and the tool takes care of the transformations. Hence, it reduces lot of effort. Two types of transformations, loop embedding and loop extraction is proposed that helps in improving performance. PED uses the incremental analysis approach as it carries out the data dependency analysis multiple times by the user. Participation of user in parallelization decision increases the accuracy of this system. However, it requires user to understand certain concept of parallelization and data dependency.

4.2.2 Object Based Parallel Programming Assist

Most of the work in automatic or incremental programming concentrates on the C like languages and these techniques are not applicable for object-based languages. One of the premier works by Hvannberg and Krishnamoorthy[38]on interactive parallelization focuses on the object oriented code at the time of program development.

Object based parallel programming assistant targets the problems faced by programmers during development of object based parallel programs. The tool gives the programmer an opportunity to increase the parallelization by utilizing its intelligence as well as the programmer's knowledge about the program (interactively). This process enhances user's knowledge about parallel programming (serves as a trainer). Object as defined by [26] is something that consists of data and modules that can operate on the data. The usual way of writing object-based program is to define the class and create the object to invoke functions within the class. This process is invariably sequential but the assistant frees users to follow the steps and object can be used without completely defining it. Assistant partitions the program to enable concurrent execution. Following are the three types of partitioning used in assistant

- Methods partitioning – Partition potentially parallelization methods within a class.
- Object partitioning – Partitioning the function calls within the function called by one object.
- Task partitioning – Partitioning the statements like loops and blocks that can be executed in parallel



```
Class CIA{
FunctionA1()
FunctionA2()};
```

```
Class CIB{
FunctionB1()
FunctionB2()};
```

```
Class A:FunctionA1{
CIB Objb;
Objb.FunctionB1()
Objb.FunctionB2()}
```

```
Class B:FunctionB1{
Int i=0;
Sum = 0;
CIA ObjA;
for(i=0; i< 2000;i++){
Sum +=I;
ObjA.FunctionA1()}}
```

Fig9: Partitioning in parallel programming assistant based on objects

As shown in figure 9, when Class A is being defined, the assistant checks whether FunctionA1 () and FunctionA2 () are independent or not. This process repeats for all the classes like class B. The assistant changes the partitioning based on the added dependency. At the end, it prepares the parallel blocks. FunctionA1 calls classes from object B and by prior partitioning of class B functions, functionA1 can again be partitioned into one or more parallel blocks. This is known as object partitioning. Finally, FunctionB1 defines one loop and as the statements of loops are entered, the assistant figures out whether the addition of statements is hindering parallelization or not. For example, first statement within the loop is clearly iteration independent. As soon as second statement with function call from class A is added, assistant needs to confirm whether this statement is affecting the parallelism of loop or not. In case it is affecting the parallelism, assistant suggests the possible transformations that can convert this loop to a parallel block again.

This tool is a very good step towards assisting programmers for parallel program development. It takes extra burden from programmer to check dependencies and provides early information about statements that reduces the degree of parallelism in the program.

4.2.3 Animation Choreographer

Animation Choreographer [40] is a tool to visualize the parallel program execution and provides feasible alternatives of program execution to increase the temporal perspectives of the parallel program. Animation Choreographer is one of the features of PARADE (PARrallel Animation Development Environment).

4.2.4 DEEP Development Environment

DEEP [13] is a development environment that consists of set of tools for parallel programming. The tools include editor, analyzer, and debugger to assist in parallel program development. DEEP supports High Performance FORTRAN (HPF) for developing data parallel programs using MPI and FORTRAN and C programs for shared memory architecture. DEEP programming environment contains configurable panels (similar to window) where, each panel contains viewers to provide the static and dynamic information about the program. DEEP program view provides both static and dynamic information about the program. Static information includes information about number of variables, functions, parallel loops and some optimization information. However, dynamic information provides the call graph, number of loops, loop count, profiling information etc. On top level, the information provided is in compact form. However, clicking on the field of interest provides detailed information that can help in parallel program analysis and design. DEEP also provides a unique way to represent program graphically. Separate rectangle represents each module in the program and each pixel in the rectangle represents individual line of program. The graphical view also provides indentation in lines to account for loops, conditional blocks etc. This whole view of program also uses color codes where colors vary from blue to red. Red color symbolizes the message-passing requirement and unsuitability for parallelism and blue symbolizes the parallel code segment. DEEP also provides a load balancing display to generate information whether processor is utilized for message passing frequency or for processing. Apart from above mentioned specific features, DEEP also provides generic features like code abstraction viewer, symbol viewer and performance viewer that provides detailed information about the program flow, data locality and performance of the program.

4.2.5 PTP-PLDT

PTP and PLDT tools, [24] developed by IBM to provide parallel programming assistance tools in eclipse environment. Assistance tools such as hover and content assistant aims to identify artifacts in parallel program developed using MPI, Open MP, and LAPI. Static-analysistools are used to performance Open MP concurrency analysis and MPI barrier analysis to detect deadlocks.

4.2.6 GRED

GRED [14] is the graphical editor for graphical programming environment GRADE (Graphical Application Development Environment). GRADE aims to provide easy to use and effective tool to develop general message passing application for heterogeneous architecture. GRED editor is used to develop application using GRAPNEL programming language that is based on message passing paradigm. The program development in GRAPNEL becomes easy by use of GRED. Every process is defined graphically as box in the editor. These processes are kept under same group as one single unit, whenever similar message passing is required. The program design is divided into three levels. At top most level, program all the components are drawn along with the interaction among them. Middle level design focuses on the message passing requirements. Lowest level design focuses low-level codes for each process. Based on these levels, GRED offers three windows called application window, process window and text editor window respectively. In short, this tool hides the lower level abstractions about parallel programming from programmers. This assumes that the programmer has a minimum expertise in parallel programming and the target is to train such programmer through the visual program development approach adapted by GRED.

4.2.7 VISO

Visual Occam (VISO) is a visual programming language for parallel or concurrent programming. VISO uses the graphical syntax based on Occam language. Semantics of VISO is represented in petri net and process calculus. VISO creates processes that have no shared data. Communication among these processes happens by the use of message passing. There are three abstraction levels and separate window represents each level. The three levels are known as system, process, and statement. System window represents all theprocess and communication among these processes. Process window uses separate window to show process and the statements that each process uses. Statement window eventually shows the details of statements which is used in processes.

4.2.7 The SUIF compiler

The SUIF compiler [25] is first of its kind, automatic sequential to parallel code conversion tool for C and FORTRAN language. It was developed to automatically convert sequential dense matrix computations, written in C or FORTRAN, to parallel code for machines with shared memory. The SUIF compiler includesmultiple optimizations passes for program analysis. The analysis includes symbolic analysis, parallelism and locality analysis, communication and synchronization analysis and code generation.

5. EASYPAR

This section describes our ongoing work on the parallel program assistance tool called EasyPar[9], [10]. This tool is named as “EASYPAR” – a combination of EASY development of PARallel codes. As the name suggests, this tool eases parallel programming by providing assistance during the development of program. Figure 11 shows the working methodology of EasyPar. EasyPar consists of two major components, first an IDE (Integrated Development Environment) and second, an Intelligent analysis engine to detect dependency between code segments which is under development. IDE provides a window to write serial code and shows vital information about concurrency to the user. Additionally, it does all the analysis required for automatic parallelization and suggests that code that can be executed on different cores. It is an interactive tool that takes input from user about the program as all the information is not available during static analysis. Such information from user is very much helpful in taking parallelization related decision. Finally, developer gets a concurrent program with different segments segregated on to different cores along with the inserted synchronization constructs. The static code analysis is the heart of any automatic parallelization technique. Automatic parallelization of the code is an old research area and many researchers have published their benchmarking work in this area. We have already discussed a number of such tools available in literature or available commercially. There are many techniques that propose the analysis of code during development like incremental parsing [21], incremental dependency analysis [53],[54],[55], incremental profiling [19],[41],[16], incremental flow graph analysis [17],[42], and so on. Still, identifying and updating data dependency information dynamically, while the code is in development phase poses many challenges. In addition, the analysis is performed in the background when programmer is developing the code and he/she wants to be unperturbed due to the background analysis. EasyPar attempts to solve this problem by employing two novel techniques.

5.1 Parallel data dependency analysis

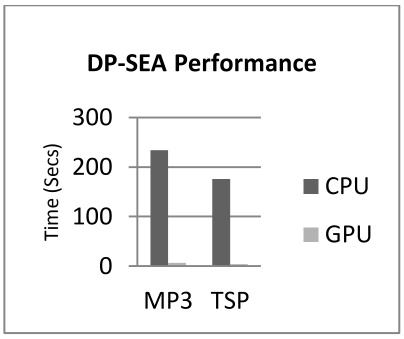


Fig 10: Strategies to handle the real time performance (Green blocks)

Static program analysis primarily includes the side effect analysis (SEA) [57], alias analysis [56] and loop analysis [81]. Alias analysis and SEA consumes most of the time due to its iterative nature. Most of the compilers use very lightweight conservative algorithm to reduce analysis time. The exhaustive SEA is computationally expensive and makes it less practical. SAE study suggests that although it is computationally intensive, redesigning it to a Data Parallel (DP) version will make it efficient. In addition, the GPGPU available in current generation desktops with massive parallel processing power is best suited for DP algorithms. We have implemented the one such SEA algorithm on GPGPU and achieved very high speed up[9]. Figure 10 shows the performance improvement of SEA algorithm on GPGPU compared to CPU. Most of the static analysis algorithms are iterative in nature. DP algorithms reduce the time of execution as well as the number of iteration [9]. Therefore, our approach enables real time analysis of code, which is required for tools such as EasyPar. To the best of our knowledge, this is the first attempt that uses GPUs and data parallel algorithm to improve the performance of data-dependency analysis algorithm.

5.3 Database based compiler

As shown in Fig 11, code written in the IDE is passedthrough a parser.Parser creates an abstract syntax tree (AST) of the code, which possesses all the vital information about the program. Dependency analysis uses AST information and accordingly identifies concurrent sections of code. We propose a significantmodificationin method for construction of AST. Usage of database management system (DBMS) concept can reduce the time for searchinginformation related to program. In addition, modification of databasebased on the search criteria is easier and faster (using database queries) as compared to data structures. This is very important improvementas compared to other automatic parallelization tools because this tool analyzes the code during development and it needs to modify the AST accordingly. To the best of our knowledge,very few of the existing compilers uses the database concept for storage and analysis of program. CAPO [102] also uses database in program analysis but it is just used to store dependence information. Database based compiler also allows scalableprogram analysis.

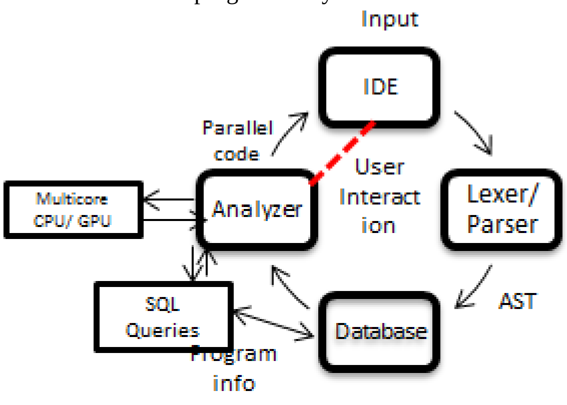


Fig 11: Strategies to handle the real time performance (Green blocks)

6. CONCLUSION

This paper discusses the issues of parallel programming, need of parallel programming tools, available tools and their limitationsWe have presented a detailed classification of the parallelization tools based on three different aspects i.e. the era of tool development, stages where the tool is useful and whether it is graphic assistance tool or not. We have explained the requirement of future parallelization tools based on the explanation of existing tools and future requirements. Easy par is a step towards development of a tool that can overcome the challenges posed by existing tools. The biggest challenge in development of such assistance tool is the performance of dependency analysis algorithms. Asthe program analysis runs intermittently, it is very important that user is unperturbed because of this analysis. Otherwise, it would become frustrating to the user. We have proposed two techniques to overcome the time complexity of dependence analysis. The first approach is to design parallel dependency algorithm running on GPU. Another approach is to create a database instead of data structures. This allows incremental, faster, and scalable concurrency analysis. It is evident that a number of tools are available that help development of parallel programs. Some of them convert existing tools to parallel code and some provide parallel programming techniques for developing new parallel programs. However, very few toolssupport programmers during the development of a program. Tools such as EasyPar, Kremlin, Kismet, S2P would be very important to increase the parallel programming capabilities among future programmers.

7. ACKNOWLEDGMENTS

Our sincere thanks to CREST team at KPIT Cummins for their constant support and ideas to improve the work in this paper.

8. REFERENCES

[1] Official home page Message Passing Interface (MPI) Available : www.mpi-forum.org/

[2] official home page - Open MP, Available : www.openmp.org

[3] Sutter, H., “The free lunch is over,” Dr. Dobb’s Journal, vol. 30, March 2005

[4] Per Stenstrom, The Paradigm Shift to MultiCores: Opportunities and Challenges, Appl. Comput. Math. (2007) 253-257

[5] VivekSarkar, Programming challenges for multicore Parallel systems, Presentation for Computer Science Department, Rice University. http://www.rice.edu

[6] McKinsley, Karthryn S. Automatic and Interactive Parallelization, PhD. Thesis, Computer Science, Rice University. Houston, Texas (1994) 12-32.

[7] Tournavitis, G. and Franke, B., “Semi-automatic extraction and exploitation of hierarchical pipeline parallelism using profiling information,” in Proceedings of the 19th international conference on Parallel architectures and compilation techniques, PACT ’10, (New York, NY, USA), pp. 377–388, ACM, 2010.

[8] Minjang Kim, dynamic program analysis algorithms to assist parallelization, PhD. thesis proposal, Georgia Institute of Technology, 2011. pp 5-28.

[9] S. Sah and VinayG. Vaidya, Paradyn: A Dynamic Parallel Programming Tool, ICDCN, 12th International Conference on Distributed Computing and Networking, PhD forum (2010)

[10] ParMA - Parallel Programming for Multi-core Architectures, Available : http://www.parma-itea2.org

[11] Par4All homepage : URL : http://www.par4all.org

[12] Chirag Dave, HansangBae, Seung-Jai Min, Seyong Lee, Rudolf Eigenmann, and Samuel Midkiff, "cetus: a sourceto-source compiler infrastructure for multicores", Computer, Vol. 42, no. 12, pp. 36-42, Dec. 2009, doi:10.1109/MC.2009.385

[13] B.Q. Warber, C.R. Brode and F. L. Orlando, DEEP: a development environment for parallel programs., Parallel Processing Symposium, IPPS/SPDP (1998) 588-593.

[14] Peter Kacsuk, Gabor Dozsa, TiborFadgyas, Robert Lovas, “The GRED Graphical Editor for the GRADE Parallel Program Development Environment”, Budapest, Hungary : MTA-SZTAKI Computer and Automation Research Institute, Hungarian Academy of Science.

[15] G. Tournavitis, Z. Wang, B. Franke,M. O'Boyle, "Towards a holistic approach to auto-parallelization: integrating profile-driven parallelism detection and machine-learning based mapping", PLDI '09, pp 177-187.

[16] Karl Fuerlinger, Michael Gerndt, Jack Dongarra, On Using Incremental Profiling for the Performance Analysis of Shared Memory Parallel Applications, Lecture Notes in Computer Sciences, (2007) 62-71

[17] Andrew R. Bernat, Barton P. Miller, “Incremental Call-Path Profiling”, Computer Sciences Department, University of Wisconsin. Madison, WI

[18] CathalBoogerd, Leon Moonen, “On the Use of Data Flow Analysis in Static Profiling”, Software EvolutionResearch Lab, Delft University of Technology. The Netherlands

[19] “On Using Incremental Profiling for the Performance Analysis of Shared Memory Parallel Applications”, Innovative Computing Laboratory”, Department of Computer Science, University of Tennessee. Technical Report

[20] NEC IMAPCAR Technical Document – www.nec.com

[21] Graham, Wagner and Susan, Efficient and Flexible Incremental Parsing, ACM Transactions of Programming Languages and Systems, Vol. 20 (1998)

[22] Barbara G. Ryder, A Position Paper on Compiler Time Program Analysis, Computer Science Dept., Rutgers University (1997)

[23] Amdahl, Gene, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities". AFIPS Conference Proceedings, 1967, pp 483-485.

[24] Beth Tibbitts, PTP - PLDT Parallel Language Development Tools Overview, Status & Plans. Technical Report by IBM, (2007)

[25] S. P. Amarasinghe, J. M. Anderson, M. S. Lam and C. W. Tseng, An Overview of the SUIF Compiler for Scalable Parallel Machines, Seventh SIAM Conference on Parallel Processing for Scientific Computing (1995)

[26] R.J. Abbott, Integrated Approach to Software Development. John Wiley and Sons (1986)

[27] Ravichandran K.M., Bhaskar P, Annamalai. S.P. and Dr. A.P. Shanthi, Automatic Inter-procesural Parallelism, Department of Computer Science, College of Engineering, Guindy, Anna University

[28] David A. Bader, Rucheek H. Sangani , Introduction to SWARM Software and Algorithms for Running on Multicore processors, Tutorial, Georgia Institute of Technology Available- http://multicore-swarm.sourceforge.net

[29] Ashwin Kumar, Aasish Kumar Pappu, Sarath Kumar, K., SudipSanyal, “Hybrid Approach for Parallelization of Sequential Code with Function Level and Block Level Parallelization”, Parallel Computing in Electrical Engineering, (2006)

[30] W. Ambrus, A Framework for Automatic Parallelization of Sequential Programs, Proceedings of the 7th International Conference on Volume 2 (2003) 11-13

[31] M. Girkar and C. Polychronopoulos, Automatic Extraction of Functional Parallelism from Ordinary Programs, IEEE Transactions on Parallel and Distributed Systems (1992)

[32] Manish Gupta, SayakMukhopadhyay, Navin Sinha, ,Automatic Parallelization of Recursive Procedures, Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, (1999) 139 – 148

[33] R. Rugina and M. Rinard, Automatic Parallelization of Divide and Conquer Algorithms, In Proc. ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, Atlanta, GA (1999)

[34] John Hughes , Why Functional Programming Matters Institutionen for Datavetenskap, Chalmers TekniskaHogskolaGteborg, SWEDEN, Circulated as Chalmers memo (1984)

[35] NirSavit, Dan Touitou, Software Transactional Memory, ACM-PODC, Ottawa Ontario, CA, 1995

[36] Official Website : “The Multicore Association”, http:// www.multicore-association.org/

[37] NVIDIA Corporation.: NVIDIA CUDA Programming Guide. Technical Report. California. USA. (2008)

[38] E.T. Hvannberg, M.S. Krishnamoorthy, An Object-based Parallel Programming Assistant, Proceeedings of the ACM SIGPLAN Workshop on Object-Based Concurrent Programming. Vols. 24, Issue 4, (1998) 200-202

[39] Sudhakar Sah, Vinay G. Vaidya, A GPU Based Novel Design of Side Effect Analysis, ICOMEC, Goa, India, (2011) 137-143

[40] Eileen Kraemer, John T. Stasko, Towards Flexible Control of the Temporal Mapping from Concurrent Program Events to Animations. Graphics, Technical Report-Visualization and Usability Center, Georgia Institute of Technology. Atlanta, GA (1994)

[41] Karl Fuerlinger, Michael Gerndt, Jack Dongarra, On Using Incremental Profiling for the Performance Analysis of Shared Memory Parallel Applications, Lecture notes in Computer science, Springer, Berlin, Volume 4641 62-71

[42] U. Ismail, “Incremental call-graph construction for the eclipse IDE”, University of Waterloo Technical Report No. CS-2009-07, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada

[43] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, and E. Bu. “Maximizing multiprocessor performance with the SUIF compiler”, IEEE Computer, (1996)

[44] Saturnino Garcia, DonghwanJeon, Chris Louie, Sravanthi Kota Venkata, and Michael Bedford Taylor, Bridging the Parallelization Gap: Automating Parallelism Discovery and Planning, HotPar, 2nd USENIX workshop on hot topics in parallelism, Poster presentation (2010)

[45] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, A Practical Automatic Polyhedral Parallelizer and Locality Optimizer, Programming Language Design and Implementation, Vol. 43, Issue 6, (2008) http://pluto-compiler.sourceforge.net/

[46] Hannu-MattiJärvinen, MikkoTiusanen, and Antti T. x`Virtanen, Convit, a Tool for Learning Concurrent Programming, Software Systems Institute, Department of Information Science Tampere University of Technology, Finland

[47] M. Kumar. “Measuring parallelism in computation-intensive scientific/engineering applications.” IEEE TOC, Sep 1988.

[48] Muhammed S. Al-Mulhem, Concurrent programming in VISO, Concurrency : Practice and Experience Concurrency, Pract. Exper. (2000) 281–288

[49] Pong M. I-pigs: An Interactive Graphical Environment for Concurrent Programming, Computer Journal (1991) 320 330

[50] Kramer J, Magee J, Ng K, “Graphical configuration programming", Computer 1989; 22(10):53–65.

[51] V. Sarkar,“Automatic partitioning of a program dependence graph into parallel tasks”, In IBM Journal of Research and Development, pages 779–804, 1991.

[52] M. C. Rinard, D. J. Scales, and M. S. Lam. “Jade: A High-Level, Machine-Independent Language for Parallel Programming”. *IEEE Computer*, 26(6):pp 28–38, 1993.

[53] T. J. Marlowe and B. G. Ryder, “An Efficient Hybrid Algorithm for Incremental Data Flow Analysis”, In Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages. (1990) 184–196

[54] J. Yur, and B.G. Ryder, Incremental analysis of the MOD problem for C. Laboratory for Computer Science Research Technical Report LCSR-TR-254, Department of Computer Science, Rutgers University (1995)

[55] B. G. Ryder and M. C. Paull, “Incremental Data Flow Analysis Algorithms”, ACM Transactions on Programming Languages and Systems. (1988) 1–50

[56] Cooper, D. Keith, K. Kennedy, Fast Interprocedural Alias Analysis. Principles of Programming Language, Austin TX, (1989) 49-59

[57] K.D. Cooper and K. Kennedy, Interprocedural Side-effect Analysis. ACM SIGPLAN Notices, Vol 39, Issue 4, (2004) 217-228

[58] Z. Li, P.C. Yew and C. Zhu, An Efficient Data Dependence Analysis for Parallelizing Compilers, IEEE Transactions on Parallel and Distributed Systems, Volume 1, Issue 1. (1990) 26-34

[59] D. Cooper, Keith, and K. Kennedy, “Efficient Computation of Flow Insensitive Interprocedural Summary”,Proc of SIGPLAN ' 84 Symp. On Compiler Constr., Montreal, Quebec, Vol. 19, No. 6 (June 1984) 247-258

[60] Intel Parallel Advisor, Available : http://software.intel.com/en-us/articles/intel-parallel-advisor/

[61] M. Kulkarni, K. Pingali, B. Walter,G. Ramanarayanan, K. Bala, and L. P. Chew. “Optimistic Parallelism Requires Abstractions”, In *PLDI’07*, pages 211–222.

[62] C. Upson, The Application Visualiation System: A Computational Environment for Scientific Visualization, IEEECOmputer Graphics and Applications, Vol. 9 (1989) 30-42.

[63] Ronsse, M. and De Bosschere, K., “Recplay: a fully integrated practical record/replay system,” ACM Trans. Comput. Syst., vol. 17, no. 2, pp. 133–152,1999.

[64] Steffan, J. G., Colohan, C. B., Zhai, A., and Mowry, T. C., “A scalable approach to thread-level speculation,” in Proceedings of the 27th annual international symposium on Computer architecture, ISCA ’00, (New York, NY, USA), pp. 1–12, ACM, 2000.

[65] Park, S., Lu, S., and Zhou, Y., “Ctrigger: exposing atomicity violation bugs from their hiding places,” in Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, ASPLOS ’09, (New York, NY, USA), pp. 25–36, ACM, 2009.

[66] Yu, Y., Rodeheffer, T., and Chen, W., “Racetrack: efficient detection of data race conditions via adaptive tracking,” in Proceedings of the twentieth ACM symposium on Operating systems principles, SOSP ’05, (New York, NY, USA), pp. 221–234, ACM, 2005

[67] Microsoft, CHESS: Find and Reproduce Heisenbugs in Concurrent Programs. http://research.microsoft.com/en-us/projects/chess/

[68] Intel Corporation, Intel Parallel Inspector. http://software.intel.com/en-us/articles/intel-parallel-inspector/

[69] Serebryany, K. and Iskhodzhanov, T., “ThreadSanitizer: data race detection in practice,” in Proceedings of the Workshop on Binary Instrumentation and Applications, WBIA ’09, (New York, NY, USA), pp. 62–71, ACM, 2009.

[70] Microsoft, Concurrency Visualizer http://msdn.microsoft.com/en-us/library/dd537632.aspx/

[71] Intel Corporation, Intel Parallel Amplifier. http://software.intel.com/en-us/articles/intel-parallel-amplifier/

[72] Intel Corporation, Intel Parallel Advisor. http://software.intel.com/en-us/articles/intel-parallel-advisor/

[73] Prism: an analysis exploration and verification environment for software implementation and optimization on multicore architectures from CriticalBlue.http://www.criticalblue.com

[74] vfAnalyst: Analyze your sequential C code to create an optimized parallel implementation from VectorFabrics,http://www.vectorfabrics.com/

[75] Rul, S., Vandierendonck, H., and De Bosschere, K., “Extracting coarse-grain parallelism in general-purpose programs”,PPoPP ’08, (New York, NY, USA), pp. 281–282, ACM, 2008

[76] Zhang, X., Navabi, A., and Jagannathan, S., “Alchemist: A transparent dependence distance profiling infrastructure,” in Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO ’09, (Washington, DC, USA), pp. 47–58, IEEE Computer Society, 2009

[77] B. Lucas, G. D. Abraham, N. S. Collins, D. A. Epstein, D. L. Gresh, K. P. McAuliffe, An Architecture for a Scientific Visualization System, IEEE Computer Society Press Proceedings of Visualization (1992)

[78] Das, D. and Wu, P., “Experiences of using a dependence profiler to assist parallelization for multi-cores,” in IPDPS Workshops, pp. 1–8, 2010.

[79] Jeon, D., Garcia, S., Louie, C., and Taylor, M. B., "Kismet: parallel speedup estimates for serial programs," in Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications, OOPSLA '11, (New York, NY, USA), pp. 519–536, ACM, 2011.

[80] KorbinianMolitorisz, JochenSchimmel, Frank Otto, Automatic Parallelization Using AutoFutures, Automatic Parallelization Using AutoFutures, Multicore Software Engineering, Performance, and Tools, Lecture Notes in Computer Science Volume 7303, 2012, pp 78-81

[81] Moseley, T., Connors, D. A., Grunwald, D., and Peri, R., "Identifying potential parallelism via loop-centric profiling," in Proceedings of the 4th international conference on Computing frontiers, CF '07, (New York, NY, USA), pp. 143–152, ACM, 2007.

[82] Das, D. and Wu, P., "Experiences of using a dependence profiler to assist parallelization for multi-cores," in IPDPS Workshops, pp. 1–8, 2010

[83] N. R. Tallent, and J. M. Mellor Crummey. "Effective performance measurement and analysis of multithreaded applications." In PPoPP '09: Proceedings of the ACM SIGPLAN symposium on Principles and practice of parallel program- ming, 2009.

[84] T. E. Anderson, and E. D. Lazowska. "Quartz: A tool for tuning parallel program performance." In SIGMETRICS, vol. 18, 1990

[85] Y. He, C. Leiserson, and W. Leiserson. "The Cilkview Scalability Analyzer." In SPAA '10, Proceedings of the Symposium on Parallelism in Algorithms and Architectures, 2010

[86] M. Kulkarni, M. Burtscher, R. Inkulu, K. Pingali, and C. Casc¸aval. "How much parallelism is there in irregular applications?" In PPoPP '09: 2009

[87] T. Harris and K. Fraser,"Language Support for Lightweight Transactions", In *OOPSLA'03*, pages 388–402, 2003

[88] William F. Appelbe, John T. Stasko, Eileen Kraemer, "Applying Program Visualization Techniques to Aid Parallel and Distributed Program Development.", Dept. of Computer Science, Georgia Institute of Technology. Atlanta, GA, Technical Report. GIT-CC 91/31 (1993)

[89] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, and T. Lawrence. "Parallel programming with Polaris", IEEE Computer (2002)

[90] M. Kim, H. Kim, and C.-K.Luk. "SD3: A scalable approach to dynamic data-dependence profiling." Microarchitecture, IEEE/ACM International Symposium on, 2010

[91] Aditi Athavale, Priti Ranadive, M. N. Babu, Prasad Pawar, Sudhakar Sah, Vinay G. Vaidya, Chaitanya Rajguru, "Automatic Sequential to Parallel Code Conversion - The S2P Tool and Performance Analysis", Journal of Computing, GSTF, Oct 2011.

[92] Official homepage, Cilk project – MIT homepage Available : www.supertech.csail.mit.edu/cilk/

[93] Official homepage, Erlang programming language homepage, Available : www.erlang.org

[94] Official homepage, Go Programming language : Available : www.golang.org

[95] Official homepage, Haskell programming language, Avaiable : www.haskell.org

[96] Intel Corporation, Intel Threading Building Blocks. http://www.threadingbuildingblocks.org/

[97] M. W. Hall, B. R. Murphy, S. P. Amarasinghe, S. Liao, and M. S. Lam. "Interprocedural Analysis for Parallelization", In LCPC'06, pp 61–80

[98] Saturnino Garcia, DonghwanJeon, Chris Louie, Sravanthi Kota Venkata, and Michael Bedford Taylor,"Bridging the Parallelization Gap: Automating Parallelism Discovery and Planning", HotPar, 2nd USENIX workshop on hot topics in parallelism, Poster presentation, 2010.

[99] U. Banerjee, "Dependence Analysis for Supercomputing", Kluwer Academic Publishers, Norwell, MA, 1988

[100] http://www.cri.ensmp.fr/pips/, accessed Oct 2011

[101] Gabriele Jost, Haoqiang Jin, Jesus Labarta, and JuditGimenez, "Interfacing Computer Aided Parallelization and Performance Analysis", ICCS'03 International conference on Computational science, 2003 pp 181-190

[102] C. Ding, X. Shen, K. Kelsey, C. Tice, R. Huang, and C. Zhang.Software Behavior Oriented Parallelization. In PLDI'07, pages 223–234, San Diego, CA