# Measuring the Effectiveness of Task-Level Parallelism for High-Level Vision

Wilson Harvey, Dirk Kalp, Milind Tambe,
David McKeown, Allen Newell

March 27, 1989

CMU-CS-89-125

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania
15213-3890



This research was partially supported by the Air Force Office of Scientific Research, under Grant AFOSR-89-0199, and by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory Under Contract F33615-87-C-1499. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Air Force Office of Scientific Research, or of the Defense Advanced Research Projects Agency, or of the United States Government.

# Table of Contents

Abstract 1
1. Introduction 1
2. Background 2
    2.1. OPS5 2
    2.2. SPAM: A Production System Architecture For Scene Interpretation 3
3. Sources of Parallelism in Production Systems 5
    3.1. Match Parallelism 5
    3.2. Task-Level Parallelism 7
4. Implementation Methodology 8
5. SPAM/PSM Implementation 11
    5.1. SPAM/PSM Architecture 11
    5.2. Measurement Techniques 13
6. Results and Analysis 13
    6.1. The Baseline System 13
    6.2. Speed-ups due to Task-Level Parallelism 14
    6.3. Speed-ups Due to Match Parallelism 15
    6.4. Multiplicative Speed-ups 16
7. Summary and Conclusions 17
8. Future Work 18
9. Acknowledgements 18
References 19

## List of Figures

Figure 1: Aerial image of San Francisco Airport
Figure 2: Interpretation phases in SPAM.
Figure 3: Speed-ups for OPS5 on the Encore Multimax [7].
Figure 4: Levels of processing in SPAM LCC.
Figure 5: Organization of the SPAM/PSM system.
Figure 6: Speed-ups varying the number of task-level processes.
Figure 7: Speed-ups varying the number of match processes.

## List of Tables

Table 1: San Francisco Airport (log #63) 4
Table 2: Washington National Airport (log #405) 4
Table 3: NASA Ames Moffett Field (log #415) 4
Table 4: Dimensions of task-level parallelism. 8
Table 5: Average, standard deviation and coefficient of variance for SF. 10
Table 6: Average, standard deviation and coefficient of variance for DC. 10
Table 7: Average, standard deviation and coefficient of variance for MOFF. 10
Table 8: Measurements for baseline system on the datasets . (Represents the optimized, ParaOPS5-based, uniprocessor version.) 14
Table 9: Multiplicative speed-ups in SPAM/PSM for SF Level 2. Parenthesized numbers are the predicted speedups. 17

## Abstract

Large production systems (rule-based systems) continue to suffer from extremely slow execution which limits their utility in practical applications as well as in research settings. Most efforts at speeding up these systems have focused on match or knowledge-search parallelism in production systems. Though good speed-ups have been achieved in this process, the total speed-up available from this source is not sufficient to alleviate the problem of slow execution in large-scale production system implementations. Such large-scale tasks can be expected to increase as researchers develop increasingly more competent rule-based systems.

In this paper, we focus on task-level parallelism, which is obtained by a high-level decomposition of the production system. Speed-ups obtained from task-level parallelism will multiply with the speed-ups obtained from match parallelism. Our vehicle for the investigation of task-level parallelism is SPAM, a high-level vision system, implemented in a production system architecture. SPAM is a mature research system having over 600 productions, with a typical scene analysis task having between 50,000 to 400,000 production firings and an execution time of the order of 10 to 100 cpu hours.

We present a characterization of task-level parallelism in production systems and, from that, select an explicit, data-driven approach for exploiting task-level parallelism. We describe a methodology for applying the chosen approach to obtain a parallel task decomposition of SPAM and to arrive at our parallel implementation, SPAM/PSM. We present the results of that implementation that show near linear speed-ups of over 12 fold using 14 processors and that point the way to substantial speed-ups from task-level parallelism.

## 1. Introduction

Large production systems (rule-based systems) continue to suffer from extremely slow execution which limits their utility in practical applications as well as research settings. Most efforts at speeding up these systems have focused on match, i.e., knowledge-search, parallelism in production systems [3, 5, 7, 15, 20, 21]. Though good speed-ups have been achieved in this process, the total speed-up available from this source is limited. Therefore, match parallelism alone will not alleviate the problem of slow execution in production systems.

In this paper, we focus on task-level parallelism, which is obtained by a high-level decomposition of the production system. Speed-ups obtained from task-level parallelism will multiply with the speed-ups obtained from match parallelism. Our vehicle for the investigation of task-level parallelism is SPAM [12, 13, 14], a high-level vision system, implemented in a production system architecture. SPAM is a mature research system having over 600 productions, with a typical scene analysis task requiring between 50,000 to 400,000 production firings and an execution time of the order of 10 to 100 cpu hours[1]. Unlike most other production systems examined for studies in parallelism, it has embedded in it a large computational demand related to the vision task that it performs. This task-related computation is separate from the computation performed for knowledge-search in the system. This is evident in the large RHS processing time for this system. While many production systems spend up to 90% of their time in knowledge-search, SPAM spends only about 30-50% of its time there.

[1]These measurements are taken from the Lisp-based version of OPS5 running on a VAX/785 processor.

In this paper, we show that the opportunities for task-level parallelism in SPAM are high and provide a much larger payoff in speed-up than match parallelism. We present a methodology and a set of principles to arrive at a suitable parallel decomposition of the SPAM task that results in near linear speed-ups of over 12 fold using 14 processors on a 16-processor shared-memory multiprocessor. Our results also indicate that a potential speed-up of 50 to 100 fold may be achievable due to task-level parallelism. We further show that match parallelism, when used in conjunction with task-level parallelism, gives another multiplicative factor of speed-up which is proportional to the size of the match component in the overall execution time. In the SPAM system, this additional multiplicative factor is around 1.5 to 2.

This paper is organized as follows: Section 2 provides some background about production systems and SPAM, the image interpretation system that is the focus of our analysis of task-level parallelism. Section 3 discusses match parallelism and task-level parallelism in production systems. We describe a new organization to compare previous work in task-level parallelism along several independent dimensions. Section 4 discusses the implementation methodology used to determine appropriate levels for task-level parallelism. We also describe a set of experiments and measurements on SPAM that allowed us to select an appropriate grain of decomposition. These techniques should be applicable to the analysis of other large production systems for evaluating the opportunities for task-level parallelism.

A new system, SPAM/PSM, resulted from the application of this methodology and its implementation is described in Section 5. Section 6 presents a detailed analysis of the results of experiments across several dimensions including grain of decomposition, speed-ups due to processor allocation for match-level and task-level parallelism. Finally, Section 7 presents a summary of our research results and Section 8 discusses some issues for future work.

## 2. Background

In this section we provide a brief overview of OPS5 and SPAM. SPAM is implemented in OPS5, hence the description of OPS5 will be useful in understanding some of the issues in how SPAM represents knowledge about spatial and structural constraints used in computer vision. Besides providing background information, this section introduces the terminology that will be used in the rest of this paper.

### 2.1. OPS5

An OPS5 [2] production system is composed of a set of *if-then* rules, called *productions*, that make up the *production memory*, and a database of temporary data structures, called the *working memory*. The individual data structures are called working memory elements (WMEs), and are lists of attribute-value pairs. Each production consists of a conjunction of condition elements (CEs) corresponding to the *if* part of the rule (also called the left-hand side or LHS), and a set of actions corresponding to the *then* part of the rule (also called the right-hand side or RHS).

The CEs in a production consist of attribute-value tests, where some attributes may contain variables as values. The attribute-value tests of a CE must all be matched by a WME for the CE to match; the variables in the condition element may match any value, but if the variable occurs in more than one CE of a production, then all occurrences of the variable must match identical values. When all the CEs of a production are matched, the production is satisfied, and an instantiation of the production (a list of WMEs that matched it), is created and entered into the *conflict set*. The

production system uses a selection procedure called *conflict-resolution* to choose a production from the conflict set, which is then *fired*. When a production fires, the RHS actions associated with that production are executed. The RHS actions can add, remove or modify WMEs, or perform I/O.

The production system is executed by an interpreter that repeatedly cycles through three steps:

1. Match
2. Conflict-resolution
3. Act

The matching procedure determines the set of satisfied productions, the conflict-resolution procedure selects a single instantiation, and the act procedure executes its RHS. These three steps are collectively called the *recognize-act cycle*.

## 2.2. SPAM: A Production System Architecture For Scene Interpretation

SPAM [12, 13, 11] is a production system architecture for the interpretation of aerial imagery with applications to automated cartography and digital mapping. It tests the hypothesis that the interpretation of aerial imagery requires substantial knowledge about the scene under consideration. Knowledge about the type of scene — airport, suburban housing development, urban city — aids in low-level and intermediate level image analysis, and will drive high-level interpretation by constraining search for plausible consistent scene models. SPAM has been applied in two task areas: airport and suburban house scene analysis. In the remainder of this section we describe the SPAM architecture, and give run-time statistics that lead us to focus on one phase for our studies in parallelism.

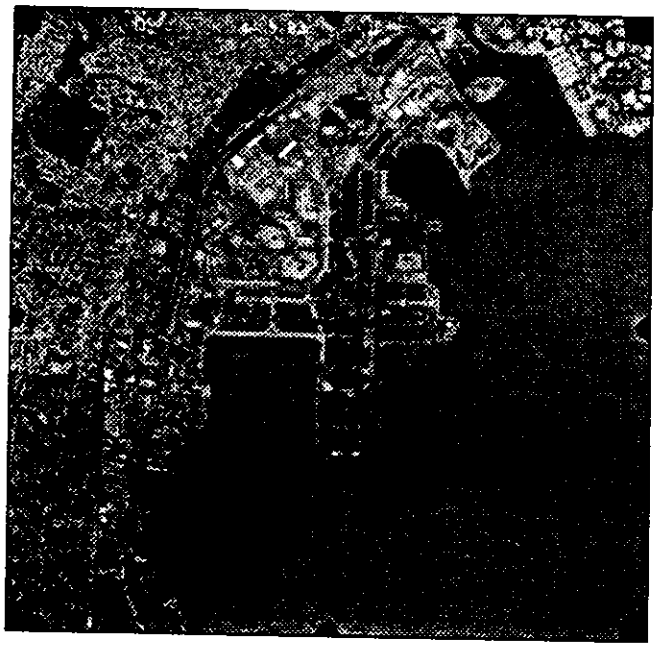

**Figure 1:** Aerial image of San Francisco Airport

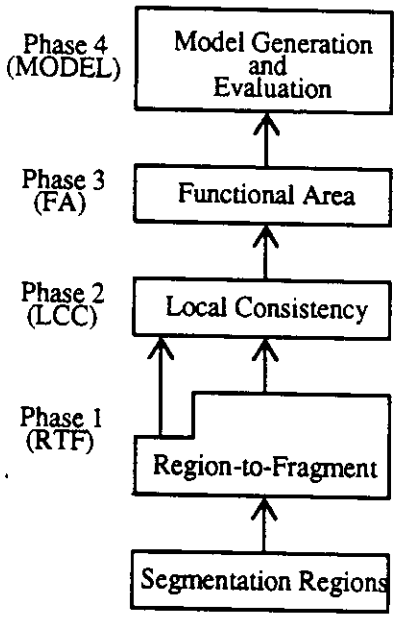


**Figure 2:** Interpretation phases in SPAM.

As with many vision systems, SPAM attempts to interpret the 2-dimensional image of a 3-dimensional scene. A typical input image is shown in Figure 1. The particular goal of the SPAM system is to interpret an image segmentation, composed of image regions, as a collection of real-world objects. For example, the output for the image in Figure 1 would be a model of the airport

scene, describing where the runway, taxiways, terminal-building(s), etc., are all located. SPAM uses four basic types of scene interpretation primitives: *regions*, *fragments*, *functional areas*, and *models*. SPAM performs scene interpretation by transforming image *regions* into scene *fragment* interpretations. It then aggregates these fragments into consistent and compatible collections called *functional areas*. Finally, it selects sets of functional areas to form *models* of the scene.

As shown in Figure 2, each interpretation phase is executed in the order given. SPAM drives from a local, low-level set of interpretations to a more global, high-level, scene interpretation. There is a set of hard-wired productions for each phase that control the order of rule executions, the forking of processes, and other domain-independent tasks. However, this "bottom-up" organization does not preclude interactions between phases. For example, prediction of a fragment interpretation in *functional-area (FA)* phase will automatically cause SPAM to reenter *local-consistency check (LCC)* phase for that fragment. Other forms of top-down activity include stereo verification to disambiguate conflicting hypotheses in *model-generation (MODEL)* phase and to perform linear alignment in *region-to-fragment (RTF)* phase.

| SPAM Phase | RTF | LCC | FA | MODEL | Total |
|---|---|---|---|---|---|
| Total CPU Time (hours) | 1.5 | 144.5 | 7.3 | 0.71 | 154.01 |
| Total Productions Fired | 11274 | 185950 | 10447 | 3085 | 210756 |
| Effective Productions/Second | 2.08 | 0.357 | 0.397 | 1.20 | 0.380 |
| Total Hypotheses | 466 | N/A | 44 | 1 | N/A |

**Table 1:** San Francisco Airport (log #63)

| SPAM Phase | RTF | LCC | FA | MODEL | Total |
|---|---|---|---|---|---|
| Total CPU Time (hours) | 2.5 | 17.9 | 7.3 | 0.33 | 28.03 |
| Total Productions Fired | 18319 | 32751 | 1483 | 1516 | 54069 |
| Effective Productions/Second | 2.03 | 0.508 | 0.056 | 1.27 | 0.536 |
| Total Hypotheses | 247 | N/A | 57 | 1 | N/A |

**Table 2:** Washington National Airport (log #405)

| SPAM Phase | RTF | LCC | FA | MODEL | Total |
|---|---|---|---|---|---|
| Total CPU Time (hours) | 0.25 | 4.12 | 2.33 | 0.33 | 7.03 |
| Total Productions Fired | 4713 | 36949 | 1503 | 3774 | 46939 |
| Effective Productions/Second | 5.24 | 2.30 | 0.160 | 3.02 | 1.85 |
| Total Hypotheses | 199 | N/A | 27 | 1 | N/A |

**Table 3:** NASA Ames Moffett Field (log #415)

Another way to view the flow of processing in SPAM is that knowledge is used to check for consistency among hypotheses; contexts are created based on collections of consistent hypotheses, and are then used to predict missing components. A collection of hypotheses must combine to create a context from which a prediction can be made. These contexts are refinements or spatial aggregations in the scene. For example, a collection of mutually consistent runways and taxiways

might combine to generate a runway functional area. Rules that encode knowledge about runway functional areas may predict that certain sub-areas within that functional area are good candidates for finding grassy areas or tarmac regions. However, an isolated runway or taxiway hypothesis cannot directly make these predictions. In SPAM the context determines the prediction. This serves to decrease the combinatorics of hypothesis generation and to allow the system to focus on those areas with strong support at each level of the interpretation.

Tables 1, 2, and 3 give statistics for run-time and number of production firings for each interpretation phase in SPAM for each of the three airports used in this study: *San Francisco International (SF)* , *Washington National (DC)*, and *NASA Ames Moffett Field (MOFF)* . It is interesting to note that LCC and FA phases account for most of the overall time in a complete run. Further, within these phases much of the RHS evaluation is performed outside OPS5 using external processes. For example, FA spends much of its time doing RHS evaluation outside of OPS5. RTF, on the other hand, spends most of its time within the traditional OPS5 evaluation model and consumes less time than FA, even though it executes a comparable number of productions. It is also clear from these tables that the application of spatial constraints in LCC makes it by far the most expensive phase in terms of amount of time spent, number of productions, as well as number of production firings.

During the LCC phase, knowledge of the structure or layout of the task domain (i.e. airports or suburban housing developments) is used to provide spatial constraints for evaluating consistency among fragment hypotheses. For example, *runways intersect taxiways* and *terminal buildings are adjacent to parking apron* are examples of the kinds of constraints that are applied to the airport scene segmentation. It is important to assemble a large collection of such consistency knowledge since the results of these tests are used to assemble fragment hypotheses found to be mutually consistent as contexts for further interpretation within the functional area phase.

As a result of this preliminary analysis we decided to focus our initial efforts on the parallel implementation of the LCC phase. Another rationale for this approach is the observation that this phase has the largest potential for growth. If a single new scene primitive is added within the RTF phase, many constraints may be added in the LCC phase in order to describe the spatial relationships (and constraints) between each of the other primitives. For these reasons, we believe that as new knowledge is added to the existing SPAM system, the proportion of time can only increase in the LCC phase.

## 3. Sources of Parallelism in Production Systems

There are two sources of parallelism in production systems: match parallelism (*MP*) and task-level parallelism (*TLP*). In this section we first discuss existing results in match parallelism. We then discuss task-level parallelism and introduce a taxonomy for describing various approaches to achieving effective speed-ups.

### 3.1. Match Parallelism

In general, production systems spend most of their time (> 90%) in the match phase of the recognize-act cycle. This makes it imperative that we speed up the match as much as possible. In the past few years, an increasing number of researchers have explored many alternative ways to speed up the match in production systems using parallelism [3, 5, 7, 15, 17, 20, 21].

Our own efforts in speeding up the match have culminated in ParaOPS5 [7, 9], a highly optimized

C-based parallel implementation of OPS5 for shared memory multi-processors. ParaOPS5 represents our current technology for achieving match parallelism within systems such as SPAM. This implementation parallelizes the highly efficient Rete [4] match algorithm. ParaOPS5 exploits parallelism at a fine granularity: subtasks execute only about 100 instructions. ParaOPS5 has been able to provide significant speed-ups for OPS5 systems that are match-intensive. Figure 3 shows the speed-ups achieved with our current implementation for three different *match intensive* systems: Rubik, Weaver and Tourney. The speed-ups are for an implementation on the Encore Multimax and are reproduced from [7]. Though Rubik and Weaver are seen to achieve good speed-ups, the speed-up in Tourney is quite low. The speed-ups are a function of the characteristics of the productions in the production system (see [6, 7].)

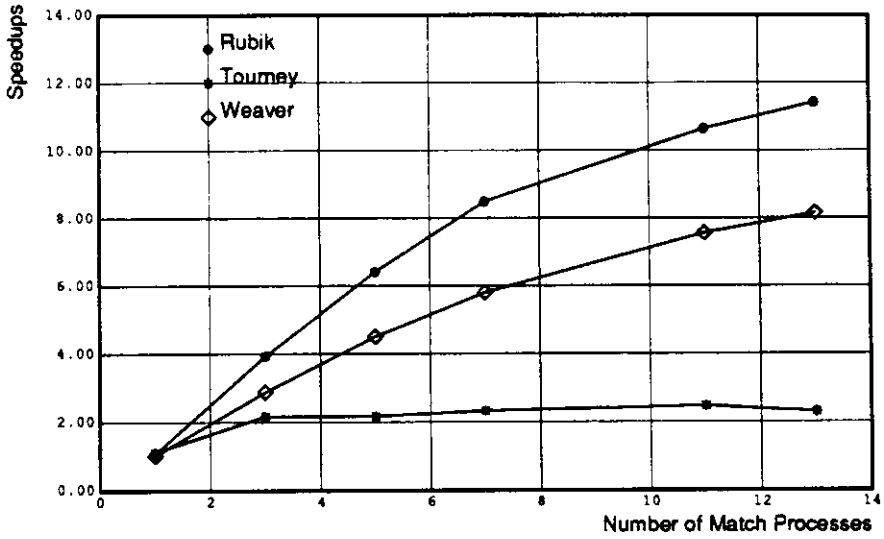


**Figure 3:** Speed-ups for OPS5 on the Encore Multimax [7].

Although systems such as ParaOPS5 have achieved good speed-ups, the total possible speed-up via MP in current production systems is limited (only 20 to 40 fold [5]). This limit is imposed by:

1. *The recognize-act cycle of OPS5*: The OPS5 model requires a synchronization in it's resolve phase. Thus MP is limited to individual cycles; we cannot extract MP across cycles.

2. *Limited match effort per cycle*: In every recognize-act cycle, only a limited number of productions are *affected*, i.e., the match effort per cycle is also quite limited.

Furthermore, MP is based on the assumption that the match phase dominates the entire computation. However, it is possible that the system under consideration is embedded in some other computationally demanding environment. In such cases, it is necessary to parallelize the rest of the computation besides match. Consider a system that spends only 50% of its time in match. Even if the match is made infinitely fast, the total speed-up possible will be only a factor of two (Amdahl's law).

## 3.2. Task-Level Parallelism

The limitations of match parallelism described in the previous section encourage the investigation of task-level parallelism. TLP has also been referred to as application parallelism [5], concept parallelism [18], and parallel rule firings [8]. The idea is to use knowledge about the problem domain to create a task decomposition suitable for parallel execution. Our choice of the term TLP for this source of parallelism is partly historical and partly dictated by the inadequacy of the other terms to cover the kind of parallelism provided by production systems like Soar [10].

A system exploiting TLP would be implemented on top of a system exploiting MP. The speed-ups obtained from these two sources can be independent and therefore multiply. We can understand TLP by considering the possible dimensions in which TLP can be divided. These dimensions are:

- *Synchronous/Asynchronous*: Synchronous production-firing systems always require a synchronization in the resolve phase of the recognize-act cycle. All the productions are matched in parallel. In the resolve phase, one or more of the productions are selected for firing. In the act phase, the selected productions are fired in parallel.

  In asynchronous production-firing systems there is no requirement for a synchronization in the resolve phase across processors. Thus, these systems do not have distinct match, resolve and act phases across the parallel system.

  Synchronous systems are less capable of handling variances in processing times for subtasks [16]. As shown in [16], given a fixed amount of work, in the presence of variance, a synchronous system quickly reaches saturation speed-ups, while an asynchronous system can continue to exploit linear speed-ups. So, in a production system embedded in an computationally intensive environment, if executing the RHS of certain productions takes much longer than others, the performance of the synchronous system will degrade heavily. However, synchronous systems may be preferred in the development and debugging stages.

- *Implicit/Explicit*: The parallelism is implicit if the system or the compiler has to *extract* parallelism out of the existing OPS5 code. This requires an analysis of the interference caused by firing productions in parallel. Thus, this is taking a dusty deck view of OPS5 programs.

  Explicit parallelism refers to providing explicit information to the system for exploiting TLP. Thus, the system may be supplied with the information that certain parts of a given task can be solved in parallel, or that certain productions can always be fired in parallel.

  In implicit parallelism, if the system engages in extracting this parallelism at compile-time, then its extraction of parallelism has to be very conservative, as the variable-bindings are unknown. If parallelism is extracted at run-time, then there are overhead costs payed at run-time. These overheads are sequential, and hence can cause considerable slowdowns. A system for exploiting explicit parallelism is able to avoid these problems.

  When the parallelism is implicit, the granularity is usually at the level of productions; it seems difficult to discover a higher level of granularity with implicit parallelism. With explicit parallelism, the user has the freedom to choose the right granularity. The level of granularity is a complex tradeoff of the number of processors available, architectural parameters, variances, data structures and task management overheads. We will discuss the granularity issue in detail in Section 4.

- *Rule distribution/working memory element distribution/No distribution*: This separation is related to the implementation of a parallel rule firing system. In the implementation of a parallel rule-firing system, it is possible to distribute the productions (rules) among processors, where each production set has its own conflict set. This distribution could be done automatically or with the help of the user. However, optimal distribution of productions among processors is a difficult problem.

  A second approach is to allocate all the productions to each processor; the working memory elements are then distributed among the processors. A third approach involves no distribution at all. Here, the parallel rule-firing is built into the control structure of the system.

Table 4 shows the various dimensions and the classification of various parallel rule-firing systems along these dimensions. These dimensions will help to investigate the TLP in SPAM/PSM. The table uses the names of authors to represent systems that do not have any names. Superscripting each system name, we indicate the third dimension that classifies the type of distribution used: rule-distribution, working memory element distribution, or none.

The SPAM/PSM system is the system described in this paper; we will discuss our design choice in detail in Section 4. These dimensions are not intended to be binary; rather, different systems could take different positions along a continuum in these dimensions. However, in the interests of clarity, the table makes a binary division. For instance, the system in [18] is classified as using implicit parallelism — however, it uses some explicit parallelism. It should be noted that except for Soar and SPAM/PSM, all other systems present simulation results on mini-production systems (with 50 or less productions).

| Dimensions | Synchronous :: Distribution | Asynchronous :: Distribution |
|---|---|---|
| Implicit | Ishida & Stolfo [8] :: Rule<br>Oshisanwo & Dasiewicz [18] :: Rule | |
| Explicit | Soar [5] [10] :: None | SPAM/PSM :: WME |

**Table 4:** Dimensions of task-level parallelism.

## 4. Implementation Methodology

In this section, we develop a methodology for applying task-level parallelism within the context of SPAM. We use knowledge about the task domain to specify several hierarchical task decompositions of the problem in which parallelism can be exploited. Thus, the characteristics of the SPAM task fit the requirements for exploiting task-level parallelism along the explicit dimension described in Section 3.2.

As described in Section 2.2, we will concentrate on the local-consistency phase (LCC) of SPAM for parallelization[2]. The LCC phase applies geometric knowledge (constraints) from the selected domain to the set of interpretations made from the dataset. This application of geometric knowledge

[2]Since the analysis is performed using the original, expensive Lisp-based SPAM system, we have extracted a representative subset of the three airport datasets to drive the analysis.

can be logically decomposed into several levels, where the tasks within each level are independent and can be performed in parallel. This is illustrated in Figure 4.

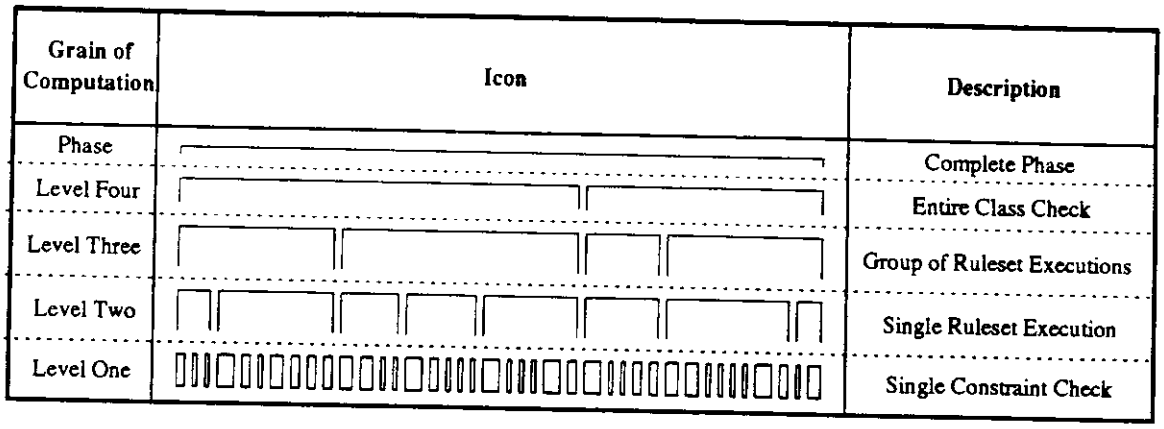


**Figure 4:** Levels of processing in SPAM LCC.

These levels of decomposition are described below:

- *LCC Phase*: At the highest phase level, the computation is for the entire LCC phase.
- *Level 4*: The phase level computation may be decomposed into tasks at Level 4, where each task applies multiple constraints to a single class of objects. For instance, a task may apply multiple constraints to all objects of class terminal building.
- *Level 3*: A single task at Level 4 may be decomposed into multiple tasks at Level 3. A task at Level 3 applies multiple constraints to a single object within the class of objects selected at Level 4. For example, a Level 3 task may apply multiple constraints to a single terminal building object.
- *Level 2*: A single task at Level 2 involves applying a single constraint to a single object. Thus, a task at Level 2 may apply a constraint such as, access roads lead to terminal buildings, to a single terminal building chosen for a task at Level 3.
- *Level 1*: A single task at Level 2 may have several components to check in applying a constraint to an object. Thus a constraint such as, access roads lead to terminal buildings, requires several roads be checked against the terminal building. A task at Level 1 would perform one of these constraint components.

Within a level, each task involves the firing of from 3 to 100 productions. As mentioned in Section 3.2, an implicit approach to extracting parallelism would make it difficult to obtain parallelism at a higher level of decomposition than individual production firings. Therefore, for this application, an explicit approach to parallelism is more appropriate.

With an explicit approach to parallelism, the choice of the right level of decomposition, or the right granularity, for parallelization must be made. This choice is determined by several factors:

1. *Task granularity*: As the average time per task gets smaller, task management overheads will have a greater impact and communication overheads and system resource contention will become more of a bottleneck.
2. *Ratio of tasks to processors*: The achievable parallelism is bounded by the number of available processors. At lower task to processor ratios, a large variance in task processing time will have a negative impact on processor utilization and the speed-ups obtained from parallelism. With higher ratios, the impact is less pronounced.
3. *Coefficient of variance*: Defined as $\sigma/\mu$, this provides a means of normalizing, for different levels of decomposition, the effect of variance in task granularity on

processor utilization. A high coefficient of variance will reduce processor utilization, resulting in lower speed-ups. This effect is more severe in synchronous systems.

4. *Decomposition effort*: This is a somewhat subjective measure. Proceeding down the hierarchy of levels, each task at the current level must be decomposed into several tasks at the next level of granularity. Usually, more work is required to specify the decomposition and design an implementation at the lower levels. The benefits of the additional parallelism that can be achieved at a lower level relative to the effort required must be assessed.

In order to choose the right level of decomposition at which to parallelize the SPAM LCC phase, we instrumented the SPAM system to obtain measurements at each level for the number of tasks and their run-time average, standard deviation, and coefficient of variance. The results of these measurements for each of the three airport datasets is presented in Tables 5, 6, and 7.

| Level | Average time per task (sec) | Standard deviation (sec) | Coefficient of variance | Number of tasks |
|---|---|---|---|---|
| Level 4 | 875.27 | 525.92 | 0.601 | 9 |
| Level 3 | 65.65 | 29.51 | 0.449 | 120 |
| Level 2 | 20.90 | 8.48 | 0.406 | 377 |
| Level 1 | 0.489 | 0.0782 | 0.159 | 16104 |

**Table 5:** Average, standard deviation and coefficient of variance for SF.

| Level | Average time per task (sec) | Standard deviation (sec) | Coefficient of variance | Number of tasks |
|---|---|---|---|---|
| Level 4 | 1308.66 | 641.72 | 0.490 | 9 |
| Level 3 | 78.51 | 30.48 | 0.388 | 150 |
| Level 2 | 24.04 | 9.51 | 0.396 | 490 |
| Level 1 | 0.430 | 0.0677 | 0.157 | 27399 |

**Table 6:** Average, standard deviation and coefficient of variance for DC.

| Level | Average time per task (sec) | Standard deviation (sec) | Coefficient of variance | Number of tasks |
|---|---|---|---|---|
| Level 4 | 165.60 | 121.20 | 0.732 | 9 |
| Level 3 | 20.07 | 8.02 | 0.399 | 74 |
| Level 2 | 5.57 | 2.43 | 0.436 | 268 |
| Level 1 | 0.349 | 0.0455 | 0.130 | 4274 |

**Table 7:** Average, standard deviation and coefficient of variance for MOFF.

Using information from Tables 5, 6, and 7, the appropriate level of granularity can now be chosen. For Level 4, the task to processor ratio is smaller than one, so we immediately rejected pursuing parallelism at this level. Levels 3 and 2 are very similar to each other in that they have enough tasks, their variances are not large, and the task granularities are much larger than the expected task management and communication overheads. Both levels, therefore, seemed to us to be worthwhile candidates. Level 3 seemed somewhat more desirable as less effort appeared to be required of us to achieve amounts of parallelism similar to that available in Level 2.

Level 1 was rejected for several reasons. First and most importantly, the additional effort involved in decomposing the system at the granularity of Level 1 would not allow us to achieve any more parallelism than at Level 2 or 3 because of the limitation on the number of processors. Second, the task granularity is much smaller and thus closer to the overheads for task management and communication than any of the other levels. Finally, the task to processor ratio is on the order of 1000. This can have a detrimental affect due to the initialization overhead. Our conclusion, then, was to exploit parallelism at the granularity of Levels 2 or 3.

The decomposition methodology can be summarized as follows:

- Analyze the baseline system and determine where the time is going.
- Determine if the explicit dimension of TLP (Section 3.2) is appropriate.
- Characterize the computation in terms of independent task decompositions at different granularities.
- Obtain measurements of the system characteristics for each level of decomposition.
- Analyze the measurements to select a level of decomposition for parallelization.

The second dimension of task-level parallelism addresses the issue of synchronous versus asynchronous execution. With an explicit decomposition at Level 3, there is no synchronization requirement. Furthermore, asynchronous models help in reducing the impact of variance. We therefore decided to decompose the system so as to allow the asynchronous rule-firings.

The final dimension of task-level parallelism addresses the issue of production versus working-memory partitioning. We decided to use working-memory partitioning, as this facilitates the explicit decomposition at the higher granularity.

# 5. SPAM/PSM Implementation

This section describes the SPAM/PSM system that implements the LCC phase of SPAM described in Sections 2.2 and 4. The system is built on top of the ParaOPS5 system described in Section 3.1. The SPAM/PSM system is implemented on an 16-processor Encore Multimax, a shared-memory multiprocessor based on the National Semiconductor NS32332 processor, rated at approximately 1.5 MIPS.

## 5.1. SPAM/PSM Architecture

Figure 5 gives a process hierarchy view of the SPAM/PSM system for the LCC phase. Viewed from the top level, the execution model consists of a *control process*, a set of *task processes*, and a *queue of tasks* to be executed. The size and number of tasks in the queue reflects the level of decomposition chosen for the LCC phase. The decomposition of LCC was described in Section 4.

The control process takes the output from the phase preceding SPAM's LCC phase and builds the queue of tasks. It then forks the task processes and, once they have completed all the tasks, collects from them the results that will be passed on to the next SPAM processing phase.

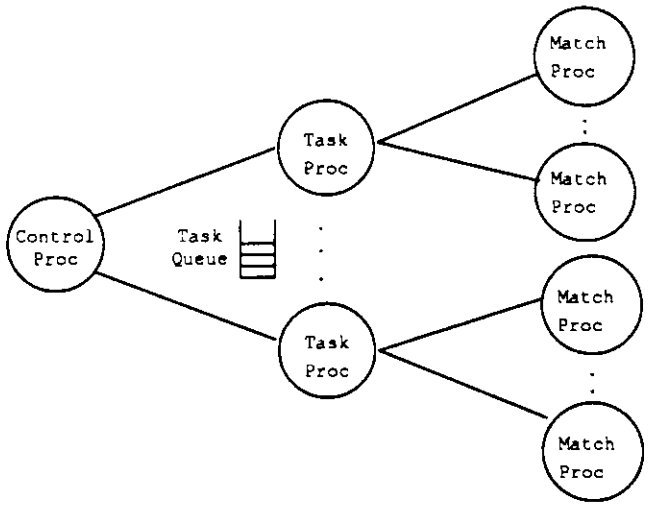


**Figure 5:** Organization of the SPAM/PSM system.

Each of the task processes is a complete and independent ParaOPS5 system. Thus, each task process has its own working memory, conflict set, Rete node memories, etc. Each task process has a production memory, which represents all the productions in the system, and effectively has a copy of the initial working memory supplied by the control process. At system initialization time, each task process can also fork a set of match processes (see Figure 5) which will perform the match in parallel.

The work performed by the SPAM/PSM system to carry out the LCC phase involves a task process removing a task from the queue and executing its ParaOPS5 system on that task. The task itself is just a WME which, when added to the process' Rete network, initializes the production system. Thus, each task can be characterized as the execution of an independent OPS5 program.

In the absence of the match processes, a task process performs the usual ParaOPS5 role of match, conflict resolution, and production firing, to carry out the OPS5 recognize-act cycle. If dedicated match processes are present, they perform the match instead, providing a second and independent axis of parallelism in the SPAM/PSM system. When there are no productions left to fire, the task is complete, and the task process goes to the queue for another task.

Thus, the SPAM/PSM system realizes our specifications:

1. *Explicit parallelism:* The decomposition of the LCC phase is explicitly specified. The task queue is initialized with independent tasks, depending on the level of decomposition, in the beginning of the run.
2. *Asynchronous production firing:* All the task processes are independent ParaOPS5 systems. Therefore, these processes can fire productions without synchronizing with each other.
3. *Working-memory element distribution:* Each task process has a copy of the entire set of productions. The working memory is distributed among the various task processes.

### 5.2. Measurement Techniques

The SPAM/PSM system is instrumented to measure the time spent in executing the tasks from two of the LCC phase decompositions, Level 2 and Level 3, identified in Section 4. The control process previously described is used to monitor and time this processing. Measurement begins at the point after which the control process has built the task queue and forked the task processes, and all the task processes have performed their initializations. Speed-ups are computed by comparing the measured execution time against the execution time of the BASELINE version, which consists of the control process, one task process, and no dedicated match processes.

Because of the 16 processor limit, we measure the effects of task-level parallelism and match parallelism in isolation. We allocate one processor for the control process, which is used only to time and not to perform tasks, and we allow one processor to the operating system. This permits us to vary the number of task processes from 1 to 14 in the isolated measurement of task-level parallelism. Next we measure the effect of match parallelism in isolation by using a single task process and varying the number of dedicated match processes from 0 to 13.

We are then able to use these two separate measures of task-level parallelism and match parallelism to predict the combined effect of the two. However, with 14 available processors, we are able to test only a subset of the possible combinations. For example, 4 task processes, each having 2 dedicated match processes, uses 12 processors (4 + (4 * 2)). Thus, dedicating 3 match processes requires 16 processors (4 + (4 * 3)) and, therefore, cannot be accommodated.

## 6. Results and Analysis

In this section we present the results of our parallel implementation, SPAM/PSM, of the SPAM LCC phase run on these three different airport datasets: SF, DC, and MOFF. As described above, the speed-ups are obtained for applying task-level parallelism and match parallelism in isolation and then for a combination of the two. We obtained results for two of the parallel decompositions, Level 3 and Level 2, identified in Section 4.

It is important to note that all the speed-ups are computed against a baseline system which represents an optimized uniprocessor implementation of the SPAM LCC phase. The original SPAM system is implemented in Lisp, using an unoptimized Lisp-based OPS5. It forks independent processes to perform geometric computations in the RHS. We ported the LCC phase of the system to C and ParaOPS5 and replaced the forked computational processes with C function calls. This baseline system itself provides approximately a 10-20 fold speed-up over the original Lisp-based implementation for the LCC phase on the three datasets used here.

### 6.1. The Baseline System

The baseline version of the system uses a single task process to execute all the tasks in the system. The results from this version are given in Table 8 and provide a picture of the magnitude of the LCC phase. The column marked DATASET gives the name of the airport and the decomposition level used. The column marked TOTAL TIME shows the total time to execute all the tasks from the queue for the given number of tasks executed. The average time per task is then shown in the next column. Finally, we further characterize the LCC phase with the total number of productions fired (PRODS FIRED), RHS actions performed (RHS ACTIONS), and changes to working memory (CHANGES TO WM).

The execution times in Table 8 provide the basis for computing all of the speed-ups. For a given

| Dataset | Total time (sec) | Number of tasks | Average time per task (sec) | Prods fired | RHS actions | Changes to WM |
|---|---|---|---|---|---|---|
| SF Level 3 | 1433 | 283 | 5.07 | 33475 | 42383 | 39116 |
| SF Level 2 | 1423 | 941 | 1.51 | 32251 | 41159 | 38550 |
| DC Level 3 | 988 | 151 | 6.55 | 20059 | 31205 | 26714 |
| DC Level 2 | 956 | 490 | 1.95 | 19418 | 30564 | 26412 |
| MOFF Level 3 | 991 | 209 | 4.74 | 22203 | 23637 | 23368 |
| MOFF Level 2 | 973 | 700 | 1.39 | 21294 | 22728 | 22950 |

**Table 8:** Measurements for baseline system on the datasets[3].
(Represents the optimized, ParaOPS5-based, uniprocessor version.)

airport dataset, there is a small difference in the total execution time between the two levels of decomposition. These differences arise due to the differences in the initial set of productions fired for generating the *tasks* for the two levels.

## 6.2. Speed-ups due to Task-Level Parallelism

The results of applying task-level parallelism are shown in Figure 6. The speed-up curves show near linear speed-ups for both levels of decomposition. The speed-ups within a level are almost the same among the three airport datasets. The maximum speed-up achieved using 14 processors is 11.90 fold in Level 3 and is 12.58 fold in Level 2.

Across the two levels, we see that the curves are consistently better in Level 2, although by only a small factor (less than 10%). While the difference is small, Level 3, with its higher granularity, was expected to have the edge in speed-up, since its task management overheads would be lower. However, the task management overheads in both levels are very low: less than .25 seconds, or less than .1% of the processing time for all the tasks. Moreover, the coefficient of variance for tasks at both levels was seen to be the same in Section 4.

Further investigation of the individual processing times of the tasks in the queue showed that there are a few tasks in each level that have execution times that are an order of magnitude larger than the average task in that level. Some of these tasks occur at the end of the task queue and create a tail-end effect in which processor utilization is low at the end of the phase. The relative disparity of these large tasks is greater within Level 3 and thus accounts for the slightly better speed-ups in Level 2.

One way to both negate this disparity and reduce the tail-end effect would be to use a separate task queue for the larger tasks and process them at the beginning of the phase. This would result in better processor utilization and thus better speed-up curves in both levels. SPAM can provide the necessary information to indentify the sizes of the tasks. This and other related issues of scheduling tasks are subjects for future work.

[3]These datasets are larger than those shown in Tables 5, 6, and 7.

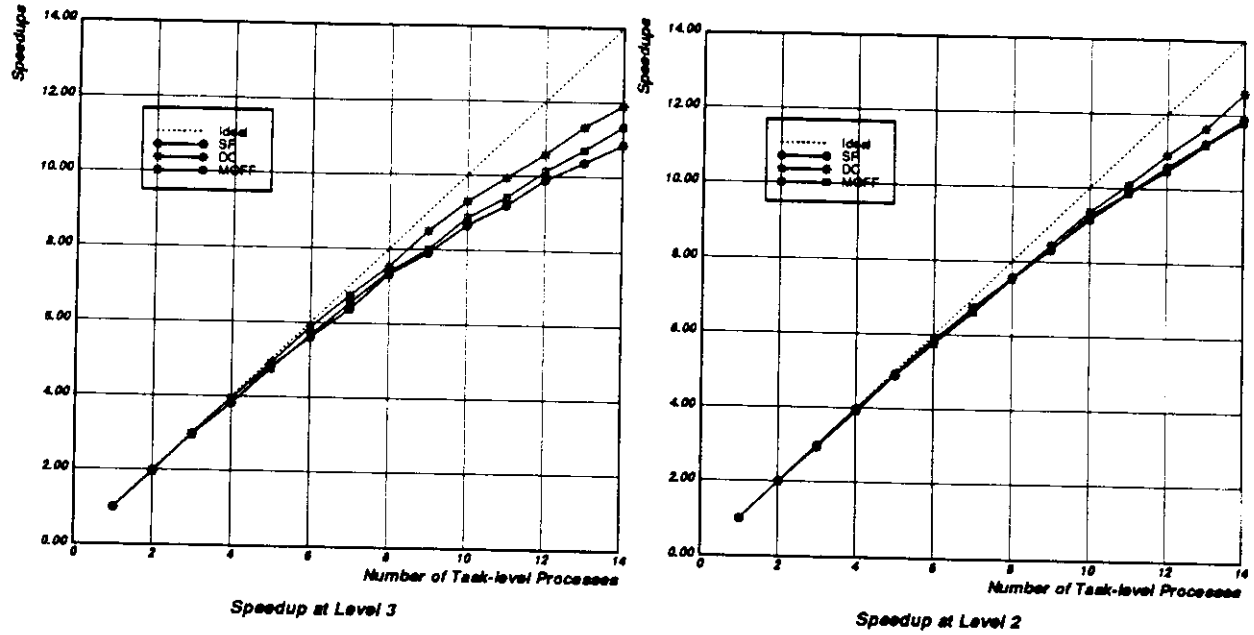


**Figure 6:** Speed-ups varying the number of task-level processes.

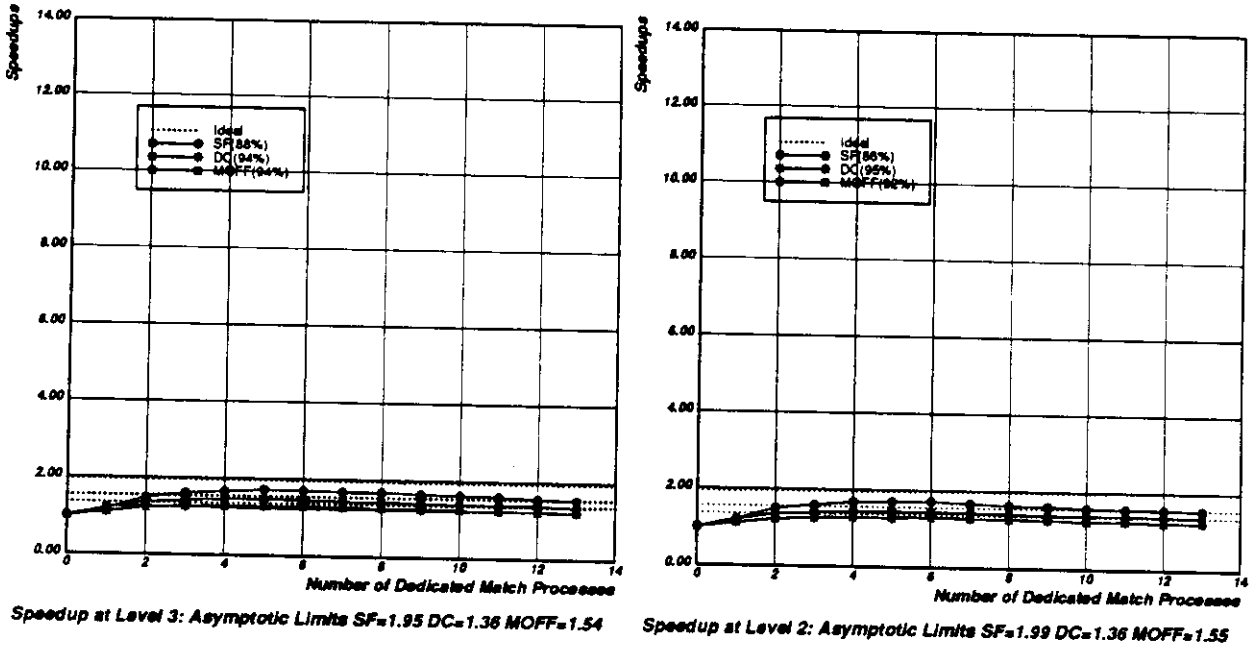


**Figure 7:** Speed-ups varying the number of match processes.

## 6.3. Speed-ups Due to Match Parallelism

Figure 7 shows the results for applying match parallelism to each of the tasks in a parallel decomposition for Levels 2 and 3. Match parallelism is obtained by dedicating processes to perform the match within the OPS5 recognize-act cycle. Since the baseline version of the system has only a task process and no dedicated match processes, it is represented in both graphs at position 0 on the horizontal axis. From the graphs, we see that applying match parallelism to the LCC phase yields very different speed-up results from those achieved using task-level parallelism. As stated in

Section 3, the theoretical maximum speed-up that can be obtained is limited according to the percentage of total execution time spent in match.

The dotted lines on the graphs show the theoretical speed-up limits. For Level 3, these limits are 1.95, 1.36, and 1.54 for SF, DC, and MOFF respectively. We were able to obtain respective speed-ups of 1.71, 1.28, and 1.45 which represent 88%, 94%, and 94% of the corresponding asymptotic limits. In all three cases, the speed-ups peaked using 6 or less match processes. Similar results are shown for Level 2.

### 6.4. Multiplicative Speed-ups

To validate the multiplicative effect of the two independent axes of parallelism [19], the system was run using task-level and match parallelism in consort. While the scope of the experiments was limited by the small number of processors, the speed-ups obtained in these combined runs were consistent with the speed-ups predicted by the multiplication of speed-ups from the two separate sources. Table 9 shows the results of some of these combined runs on SF for Level 2. The top row of the table varies the number of dedicated match processes from 0 to 5. The left column of the table varies the number of task processes from 1 to 7. The first row of numbers in the table gives the speed-ups from match parallelism in isolation. The first column of numbers in the table gives the speed-ups from task-level parallelism in isolation.

The table entry at ($Task_1$, $Match_0$) represents the baseline version of the system. Each of the other table entries shows the achieved multiplicative speed-up from the combined sources with the predicted speed-up in parentheses directly below. For example, the entry ($Task_4$, $Match_2$) represents the use of 4 task processes with each having 2 dedicated match processes. The achieved speed-up for this configuration is 5.82 fold and the predicted speed-up is 5.96 (3.98*1.50). Table entries marked with an asterisk could not be measured due to a lack of processors on the machine (see Section 5.2). For example, ($Task_4$, $Match_3$) requires 17 processors: 1 control process, 4 task-level processes, and 12 (= 4*3) dedicated match processes. The table shows the achieved speed-ups to be very close to the predicted speed-ups. Similar results were obtained for DC and MOFF.

The speed-up curves for task-level and match-level parallelism graphically indicate that the benefits from task-level parallelism are much more significant than from match parallelism. Thus, in a setting where the number of available processors is limited, it is best to allocate them to task-level parallelism rather that match parallelism. We believe that the potential for additional speed-ups in SPAM from task-level parallelism is quite high; an expectation of 50 to 100 fold does not seem unreasonable, since:

1. The tasks within any of the LCC decompositions are independent of one another.
2. Several hundred tasks are available in Level 2.
3. The task queue management overheads measured for Level 2 and Level 3 are very low, especially with respect to the task granularity, and thus are not a factor.

The current scheme of decomposition depends on a centralized task-queue for effective distribution of tasks among processes. A centralized task queue may become a bottleneck for an increasing number of processes; therefore, we need to investigate schemes for effective distribution of tasks among processes.

Though our scheme of parallelization has been presented in the context of non-match-intensive system, the scheme is applicable to match-intensive systems as well. In match-intensive systems, match parallelism will make a substantial contribution to the speed-ups.

| | $Match_0$ | $Match_1$ | $Match_2$ | $Match_3$ | $Match_4$ | $Match_5$ |
|---|---|---|---|---|---|---|
| $Task_1$ | 1 | 1.21 | 1.50 | 1.60 | 1.68 | 1.70 |
| $Task_2$ | 2.01 | 2.42 (2.43) | 2.97 (3.01) | 3.16 (3.22) | 3.30 (3.37) | 3.36 (3.42) |
| $Task_3$ | 2.98 | 3.57 (3.60) | 4.42 (4.46) | 4.73 (4.78) | * (5.01) | * (5.07) |
| $Task_4$ | 3.98 | 4.73 (4.81) | 5.82 (5.96) | * 6.37) | * (6.69) | * (6.77) |
| $Task_5$ | 4.93 | 5.82 (5.95) | * (7.39) | * (7.89) | * (8.28) | * (8.38) |
| $Task_6$ | 5.89 | 6.98 (7.12) | * (8.83) | * (9.42) | * (9.90) | * (10.01) |
| $Task_7$ | 6.70 | 8.04 (8.09) | * (10.05) | * (10.72) | * (11.26) | * (11.39) |

**Table 9:** Multiplicative speed-ups in SPAM/PSM for SF Level 2. Parenthesized numbers are the predicted speedups.

## 7. Summary and Conclusions

In this paper we characterized task-level parallelism in production systems along three dimensions and, from that, selected an explicit, data-driven, asynchronous approach for exploiting it. The system we presented, SPAM/PSM, is a real, computationally demanding, high-level vision system that relies on knowledge-based reasoning. With the SPAM/PSM system, we showed that an explicit approach to task-level parallelism can yield significant speed-ups.

The explicit approach relies on knowledge that the system designer has available about the nature of the problem. The designer uses this knowledge directly to arrive at a problem decomposition in which parallelism can be exploited. The decomposition is made based on the data upon which the system must operate and several levels of decomposition are possible. We saw that the choice of the correct level at which to exploit parallelism is based upon a number of factors; among these are the task granularity, task management and communication overheads, the variance in task processing times, and the ratio of total tasks to processors.

For the SPAM/PSM system we presented a methodology for obtaining a parallel task decomposition and arrived at three levels of decomposition. We implemented two of those levels and obtained near linear speed-ups with a maximum of over 12 fold using 14 processors. The results obtained indicate that speed-ups on the order of 50 to 100 fold from task level parallelism might be realized on a machine with a comparably large number of processors. We believe that the success achieved with the SPAM/PSM system gives hope to designers of other rule-based systems to realize systems with much lower execution times by applying task-level parallelism. Also the potential for very large speed-ups indicated here should serve as encouragement to the designers of large-scale multiprocessor systems.

We also obtained results for applying match parallelism to each of the tasks in a parallel decomposition. We saw that this speed-up represented an independent axis of parallelism and thus

could be multiplied with the speed-up obtained from task-level parallelism. In the airport data sets tested, this axis provided a factor of 1.5 to 2 fold parallelism.

We believe that the explicit, data-driven approach taken here holds better potential for realizing task-level parallelism than implicit approaches that attempt to extract parallel rule-firings using a task-independent, bottom-up analysis. With these latter kinds of approaches to task-level parellelism, there is not enough information available at compile-time to make these decisions and the complexity and overhead required at run-time to perform the analysis is prohibitive and does not seem likely to yield much speed-up from parallelism. Such parallel rule-firing schemes are still constrained by the overall synchronous nature of the OPS5 recognize-act cycle. In addition, the run-time analysis for parallel rule-firings places another synchronous constraint upon the system which presents a further bottleneck to parallelism. The top-down, explicit approach presented here achieves parallel rule-firings without this synchronous constraint and the overhead of the run-time analysis. Furthermore, the analysis required to arrive at a suitable parallel decomposition is straightforward and can be arrived at fairly quickly.

Finally, the framework for exploiting task-level parallelism presented in this paper seems most suitable for parallelizing knowledge-intensive systems that exhibit weak interaction between the individual subtasks of the task. This framework is especially useful for systems with a large computational demand separate from the demand imposed by match.

## 8. Future Work

In the near future we plan to make our SPAM/PSM implementation a useful tool for SPAM researchers. This means partitioning and parallelization of the other phases of SPAM besides the local consistency check phase. Moreover, currently, the initialization subphase within the local-consistency phase consumes a large amount of processing time. We need to optimize and/or parallelize the initialization subphase.

Results from Section 6 show that large amounts of parallelism can be exploited in SPAM, and thus, significantly larger number of processors could be employed in exploiting the parallelism. Shared-bus multi-processors like the Encore Multimax cannot support such a large number of processors. We need to evaluate other scalable parallel architectures for exploiting match and task-level parallelism in production systems. Toward this end, we are currently investigating implementations on message-passing computers; simulation results for production systems on message-passing computers [1] have shown positive results.

Our long term plan is the investigation of task-level parallelism in systems besides SPAM. We hope such investigations will help us refine the general methodology for exploiting task-level parallelism in production systems.

## 9. Acknowledgements

We'd like to thank the members of both the PSM group and Digital Mapping Lab for helpful discussions. In particular, discussions with Anurag Acharya, Lanny Forgy, and Brian Milnes did much to shape the design of the current system. Matt Diamond provided invaluable programming assistance, and Kathy Swedlow helped make our prose a bit more understandable.

## References

1. Acharya, A. and Tambe, M. Production Systems on Message Passing Computers: Simulation results and analysis. Computer Science Department, Carnegie Mellon University, December, 1988 In preparation.

2. Brownston, L., Farrell, R., Kant, E., and Martin, N.. *Programming Expert Systems in OPS5: An introduction to rule-based programming*. Addison-Wesley, Reading, Massachusetts, 1985.

3. Butler, P. L., Allen, J. D., and Bouldin, D. W. Parallel architecture for OPS5. Proceedings of the Fifteenth International Symposium on Computer Architecture, 1988, pp. 452-457.

4. Forgy, C. L. OPS5 User's Manual. Tech. Rept. CMU-CS-81-135, Computer Science Department, Carnegie Mellon University, July, 1981.

5. Gupta, A. *Parallelism in Production Systems*. Ph.D. Th., Computer Science Department, Carnegie Mellon University, March 1986. Also available in *Parallelism in Production Systems*, Morgan-Kaufman, 1987.

6. Gupta, A., Forgy, C. L., Kalp, D., Newell, A., and Tambe, M. Parallel OPS5 on the Encore Multimax. Proceedings of the International Conference on Parallel Processing, August, 1988, pp. 271-280.

7. Gupta, A., Tambe, M., Kalp, D., Forgy, C. L., and Newell, A. "Parallel implementation of OPS5 on the Encore Multiprocessor: Results and analysis". *International Journal of Parallel Programming 17*, 2 (1988). In press.

8. Ishida, T. and Stolfo, S. Towards the parallel execution of rules in production system programs. Proceedings of the International Conference on Parallel Programming, August, 1988, pp. 568-574.

9. Kalp, D., Tambe, M., Gupta, A., Forgy, C., Newell, A., Acharya, A., Milnes, B., and Swedlow, K. Parallel OPS5 User's Manual. Tech. Rept. CMU-CS-88-187, Computer Science Department, Carnegie Mellon University, November, 1988.

10. Laird, J.E., Newell, A. and Rosenbloom, P.S. "Soar: An architecture for general intelligence". *Artificial Intelligence 33*, 1 (1987), 1-64.

11. McKeown, D.M., Harvey, W.A., Wixson, L. "Automating Knowledge Acquisition For Aerial Image Interpretation". *Computer Vision, Graphics and Image Processing 46*, 1 (April 1989), 37-81.

12. McKeown, D.M., Harvey, W.A., and McDermott, J. "Rule based interpretation of aerial imagery". *IEEE Transactions on Pattern Analysis and Machine Intelligence 7*, 5 (1985), 570-585.

13. McKeown, D., McVay, W., and Lucas, B. "Stereo verification in aerial image analysis". *Optical Engineering 25*, 3 (1986), 333-346.

14. McKeown, D., Harvey, W., and Wixson, L. Automating knowledge acquisition for aerial image interpretation. To appear in *Computer Vision, Graphics and Image Processing*.

15. Miranker, D. P. Treat: A better match algorithm for AI production systems. Proceedings of the National Conference on Artificial Intelligence, August, 1987, pp. 42-47.

16. Mohan, J. *Performance of Parallel Programs: Model and Analyses*. Ph.D. Th., Computer Science Department, Carnegie Mellon University, July 1984.

**17.** Oflazer, K. *Partitioning in Parallel Processing of Production Systems*. Ph.D. Th., Computer Science Department, Carnegie Mellon University, March 1987.

**18.** Oshisanwo, A. and Dasiewicz, P. A parellel model and architecture for production systems. Proceedings of the International Conference on Parallel Processing, August, 1987, pp. 147-153.

**19.** Reddy, R. and Newell, A. Multiplicative speedup of systems. In Jones, A., Ed., *Perspectives on Computer Science*, Academic Press, New York, New York, 1977, pp. 183-198.

**20.** Schreiner, F. and Zimmerman, G. Pesa-1: A parallel architecture for production systems. Proceedings of the International Conference on Parallel Processing, August, 1987, pp. 166-169.

**21.** Tambe, M., Kalp, D., Gupta, A., Forgy, C.L., Milnes, B.G., and Newell, A. Soar/PSM-E: Investigating Match Parallelism in a Learning Production System. Proceedings of the ICM/SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages, and Systems, July, 1988, pp. 146-160.