Airbreathing Propulsion System Analysis Using Multithreaded Parallel Processing

Richard Gregory Schunk[1] and T. J. Chung[2]
[1]NASA/MSFC, Huntsville, Alabama
[2]Department of Mechanical & Aerospace Engineering
The University of Alabama in Huntsville

SUMMARY

In this paper, parallel processing is used to analyze the mixing and combustion behavior of hypersonic flow. Preliminary work for a sonic transverse hydrogen jet injected from a slot into a Mach 4 airstream in a two-dimensional duct combustor has been completed [Moon and Chung, 1996]. Our aim is to extend this work to three-dimensional domain using multithreaded domain decomposition parallel processing based on the flowfield-dependent variation theory [Schunk, Canabal, Heard, and Chung, 1999; Schunk and Chung, 1999].

Numerical simulations of chemically reacting flows are difficult because of the strong interactions between the turbulent hydrodynamic and chemical processes. The algorithm must provide an accurate representation of the flowfield, since unphysical flowfield calculations will lead to the faulty loss or creation of species mass fraction, or even premature ignition, which in turn alters the flowfield information. Another difficulty arises from the disparity in time scales between the flowfield and chemical reactions, which may require the use of finite rate chemistry. The situations are more complex when there is a disparity in length scales involved in turbulence. In order to cope with these complicated physical phenomena, it is our plan to utilize the flowfield-dependent variation theory mentioned above, facilitated by large eddy simulation. Undoubtedly, the proposed computation requires the most sophisticated computational strategies. The multithreaded domain decomposition parallel processing will be necessary in order to reduce both computational time and storage. Without special treatments involved in computer engineering, our attempt to analyze the airbreathing combustion appears to be difficult, if not impossible. We describe in detail the parallel processing strategy below.

Multi-threaded programming is utilized to take advantage of multiple computational elements on the host computer. Typically, a multi-threaded process will spawn multiple threads which are allocated by the operating system to the available computational elements (or processors) within the system. If more than one processor is available, the threads may execute in parallel resulting in a significant reduction in excution time. If more threads are spawned than available processors, the threads appear to execute concurrently as the operating system decides which threads execute while the others wait. One unique advantage of multi-threaded programming on shared memory multiprocessor systems is the ability to share global memory. This alleviates the need for data exchange or message passing between threads as all global memory allocated by the parent process is available to each thread. However, precautions must be taken to prevent deadlock or

race conditions resulting from multiple threads trying to simultaneously write to the same data.

Threads are implemented by linking an application to a shared library and making calls to the routines within that library. Two popular implementations are widely used: the Pthreads library (and its derivatives) that are available on most Unix operating systems and the NTthreads library that is available under Windows NT. There are differences between the two implementations, but applications can be ported from one to the other with moderate ease and many of the basic functions are similar albeit with different names and syntax.

Domain decomposition methods can be used in conjunction with multi-threaded programming to create an efficient parallel application. The sub-domains resulting from the decomposition provide a convenient division of labor for the processing elements within the host computer. In this application, an Additive Schwarz domain decomposition method is utilized. The method is illustrated below (Figure 1) for a two dimensional square mesh that is decomposed into four sub-domains. The nodes belonging to each of the four sub-domains are denoted with geometric symbols while boundary nodes are identified with bold crosses. The desire is to solve for each node implicitly within a single sub-domain. For nodes on the edge of each sub-domain this is accomplished by treating the adjacent node in the neighboring sub-domain as a boundary. The overlapping of neighboring nodes between sub-domains is illustrated in Figure 2. Higher degrees of overlapping, which may improve convergence at the expense of computation time, are also used.

In a parallel application, load balancing between processors is critical to achieving optimum performance. Ideally, if a domain could be decomposed into regions requiring an identical amount of computation, it would be a simple matter to divide the problem between processing elements as shown in Figure 3 for four threads executing on an equal number of processors.

Unfortunately, in a "real world" application the domain may not be decomposed such that the computation for each processor is balanced, resulting in lost efficiency. If the execution time required for each sub-domain is not identical, the CPU's will become idle for portions of time as shown in Figure 4.

One approach to load balancing, as implemented in this application, is to decompose the domain into more sub-domains than available processors and use threads to perform the computations within each block. The finer granularity permits a more even distribution of work amongst the available processing elements as shown in Figure 5.

In this approach, the number of threads spawned is equal to the number of available processors with each thread marching through the available sub-domains (which preferably number at least two times the number of processors), solving one at a time in an "assembly-line" fashion. A stack is employed where each thread pops the next sub-domain to be solved off of the top of the stack. Mutual exclusion locks are employed to protect the stack pointer in the event two or more threads access the stack simultaneously. Each thread remains busy until the number of sub-domains is exhausted. If the number of sub-domains is large enough, the degree of parallelism will be high although decomposing a problem into too many sub-domains may adversely affect convergence.

The above approach will be utilized for the analysis of hypersonic airbreathing combustion with hydrogen fuel. Without using the parallel processing, the analysis with Mach 4 free-stream velocity for the finite rate chemistry with 18 species has been carried out as shown in Figs. 6 through 8 [Moon and Chung, 1996]. In this study, the standard $K-\varepsilon$ model was used. The proposed paper will utilize the large eddy simulation with high Mach numbers and high Reynolds numbers. It is our plan to report on the maximum ranges of Mach number and Reynolds number the proposed algorithm can accommodate. Our eventual goal in this paper is to determine the relationship between mixing and combustion efficiency. Length scales and time scales involved in turbulence and combustion will be thoroughly investigated.

References

Moon, S.Y. and Chung, T. J. [1996]: Numerical simulation of heat transfer in chemically reacting shock wave turbulent boundary layer interactions, Journal of Numerical Heat Transfer, Part A, 30, 1,55-72.

Schunk, R. G, Canabal, F., Heard, G.A., and Chung, T.J. [1999]: Unified CFD methods via flowfield-dependent variation theory. AIAA paper 99-3715.

Schunk, R. G. and Chung, T. J. [1999]: Parallelization of the flowfield-dependent variation schemes for solving the triple shock/boundary layer interaction problem, TFAWS 99 NASA/MSFC, Septemer 13-15, 1999.

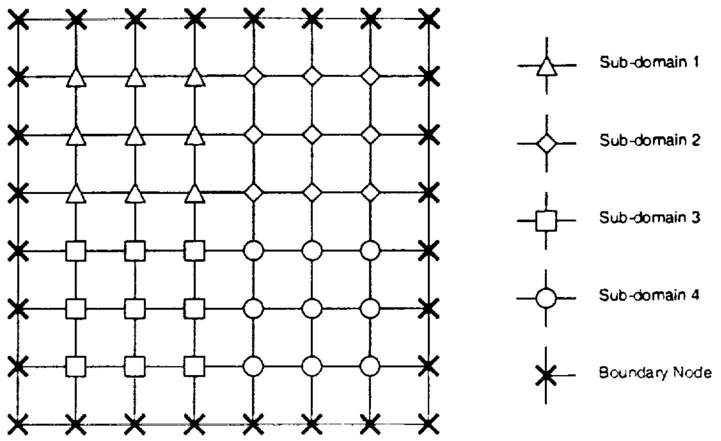


Figure 1: Multiple Subdomains

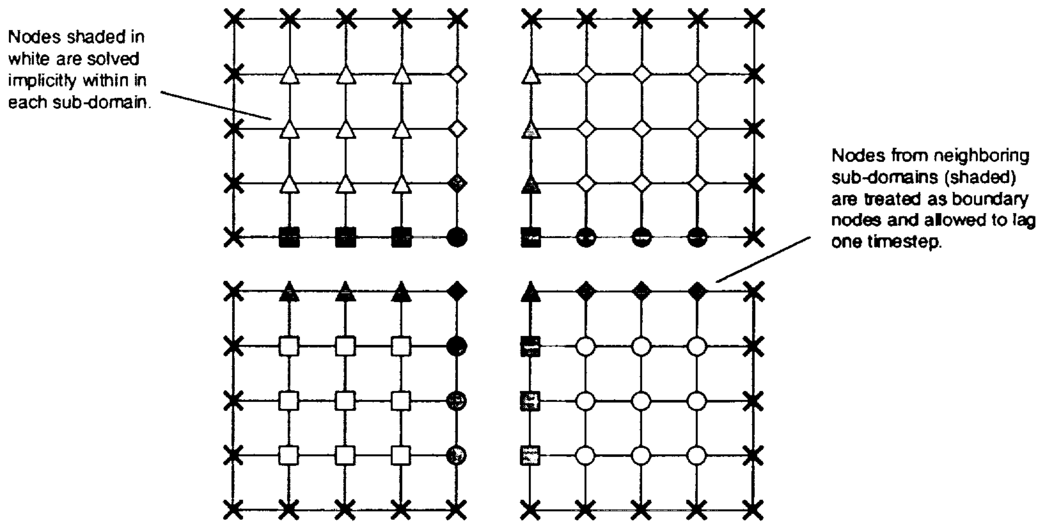

Nodes shaded in white are solved implicitly within in each sub-domain.
Nodes from neighboring sub-domains (shaded) are treated as boundary nodes and allowed to lag one timestep.

Figure 2: Domain Decomposition

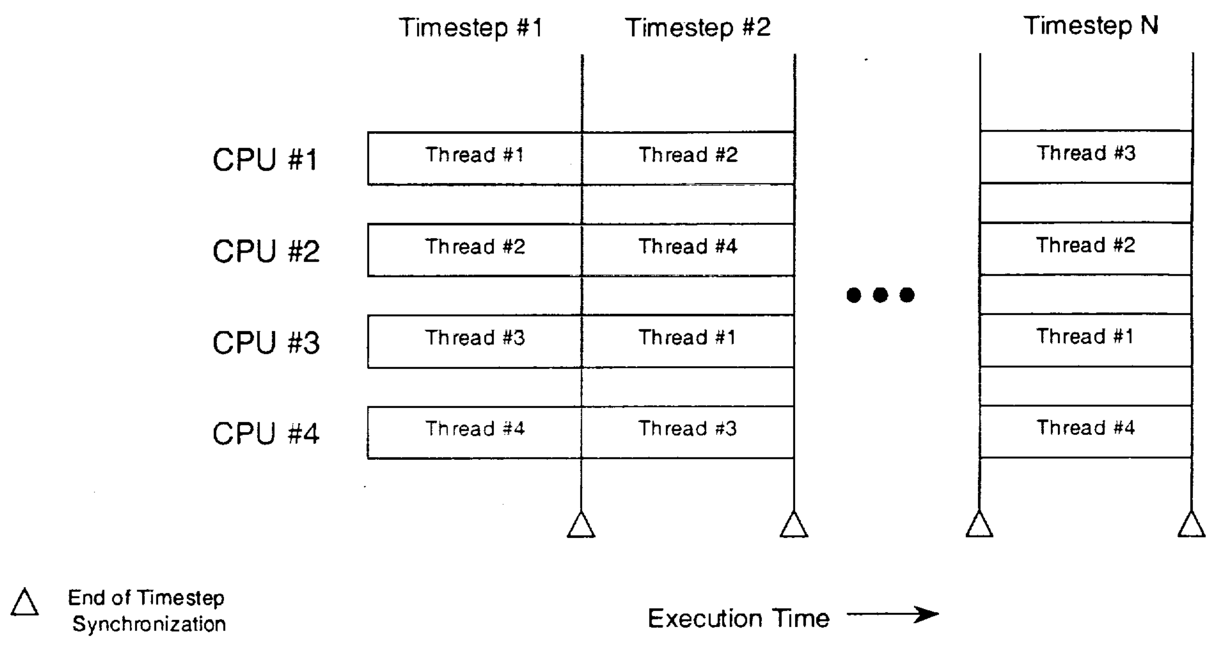


Figure 3: Ideal Load Balancing

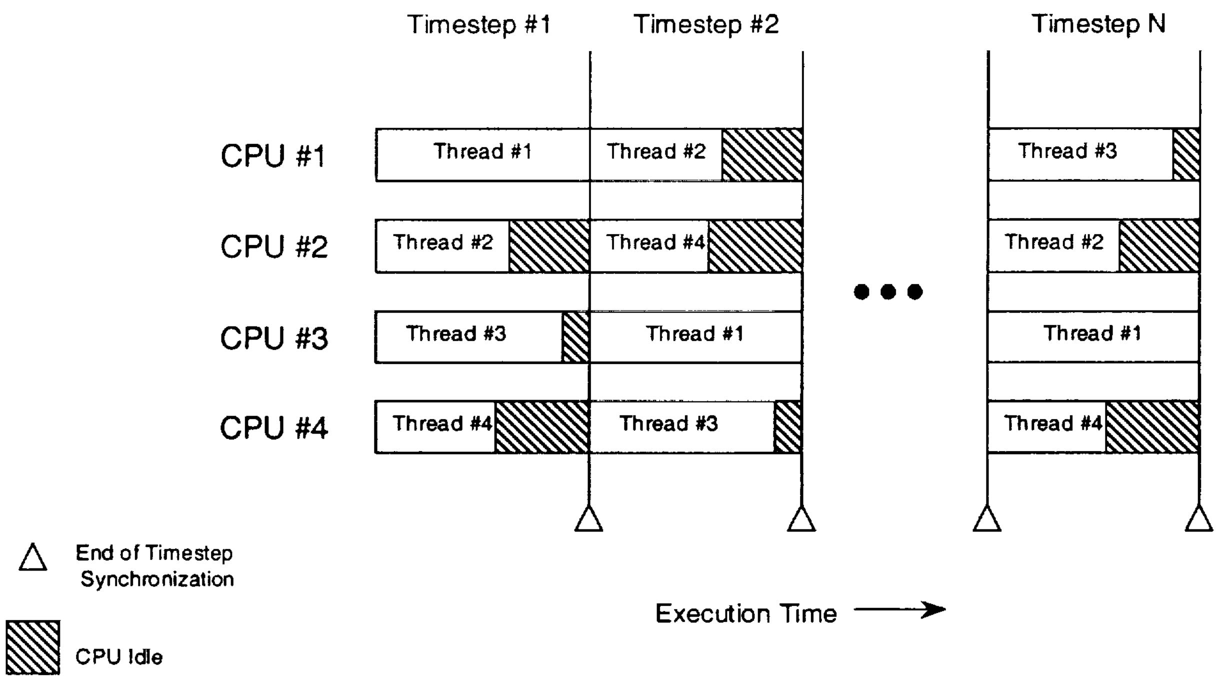


Figure 4: "Real World" Load Balancing

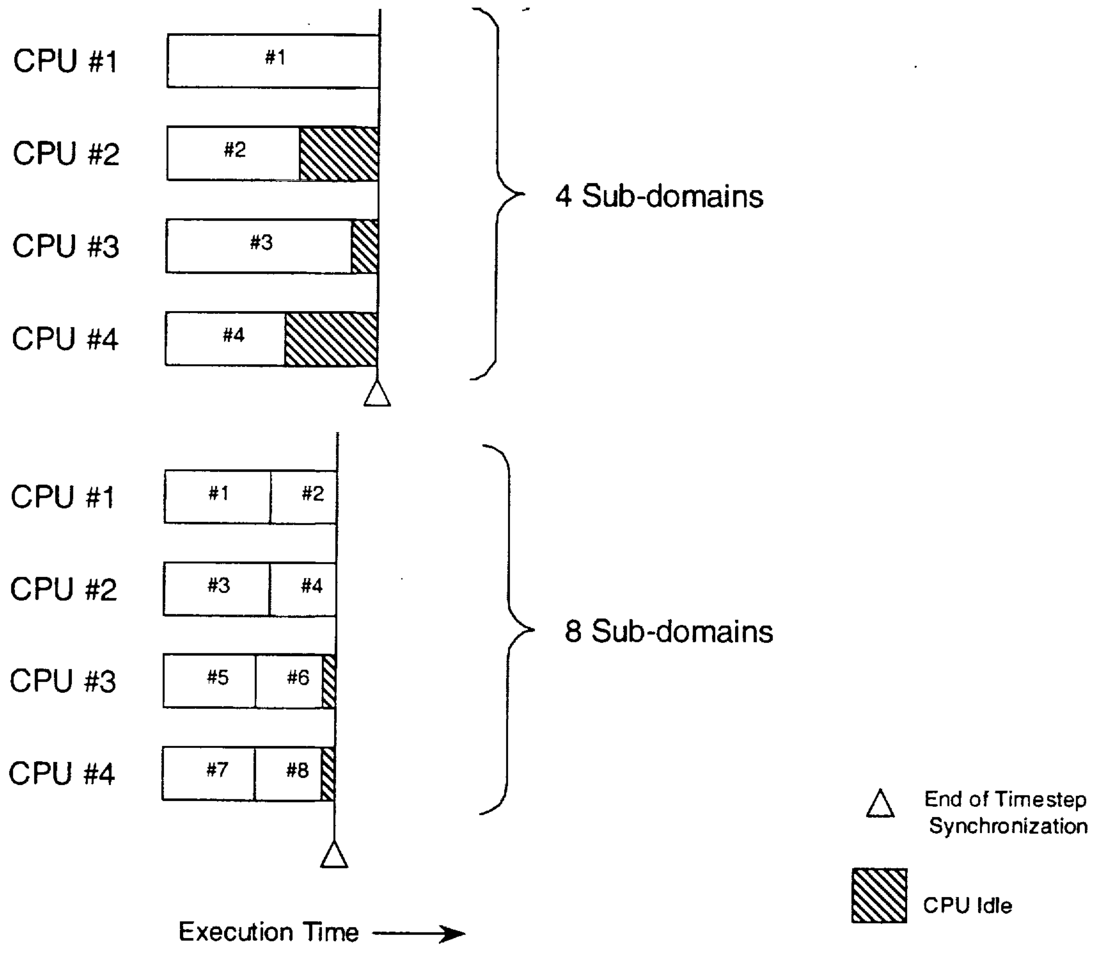


Figure 5: Domain Decomposition Improves Parallelism

Airbreathing Propulsion System Analysis Using Multi-threaded Parallel Processing

Greg Schunk
NASA/MSFC
University of Alabama in Huntsville
richard.schunk@msfc.nasa

T. J. Chung
University of Alabama in Huntsville

Introduction

- The objective of this research is to demonstrate the application of the Flow-field Dependent Variation (FDV) method to a problem of current interest in supersonic chemical combustion.

- Due in part to the stiffness of the chemical reactions and the number of constituents, the solution of such problems on unstructured three dimensional grids often demands the use of parallel computers.

- Preliminary results for the injection of a supersonic hydrogen stream into vitiated air are presented.

Flow-field Dependent Variation Approach

The conservation of mass for a chemical species, k, may be represented as follows:

$$\frac{\partial U_k}{\partial t} = B_k - \frac{\partial F_i}{\partial x_i} - \frac{\partial G_i}{\partial x_i}$$

$$U_k = [\rho Y_k]$$

$$F_i = [\rho Y_k v_i]$$

$$G_i = [-\rho D_{km} Y_{k,i}]$$

$$B_k = w_k$$

where Y_k represents the species mass fraction, F is the convective flux, G is the diffusive flux, and w_k is the generation of species k from chemical reaction

Development of the FDV Equations

Continuity Equations in Conservation Form:

$$\frac{\partial U_k}{\partial t} = B_k - \frac{\partial F_i}{\partial x_i} - \frac{\partial G_i}{\partial x_i}$$

Expand U^{n+1} in a special form of the Taylor series with respect to time between the n and n+1 timesteps:

$$\mathbf{U}_k{}^{n+1} = \mathbf{U}_k{}^{n} + \Delta t \frac{\partial \mathbf{U}_k{}^{n+s_5}}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 \mathbf{U}_k{}^{n+s_6}}{\partial t^2} \qquad ,(0 \le s_5 \le 1 \text{ and } 0 \le s_6 \le 1):$$

Define $\Delta U^{N+1} = U^{N+1} - U^N$ and recast the first and second order derivatives in terms of the fluctuations:

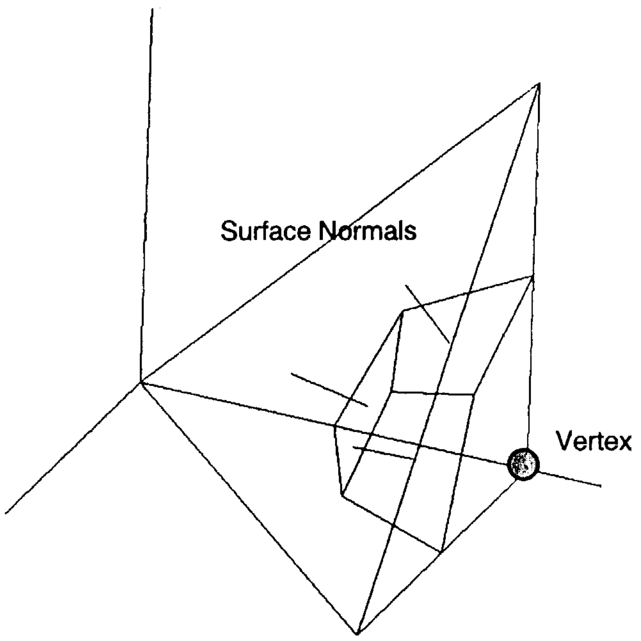
$$\frac{\partial \mathbf{U}^{n+s_5}}{\partial t} = \frac{\partial \mathbf{U}^{n}}{\partial t} + s_5 \frac{\partial \Delta \mathbf{U}^{n+1}}{\partial t} \qquad \frac{\partial^2 \mathbf{U}^{n+s_6}}{\partial t^2} = \frac{\partial^2 \mathbf{U}^{n}}{\partial t^2} + s_6 \frac{\partial^2 \Delta \mathbf{U}^{n+1}}{\partial t^2}$$

Rearrange with the fluctuation quantities on the left-hand-side and the source terms on the right-hand-side:

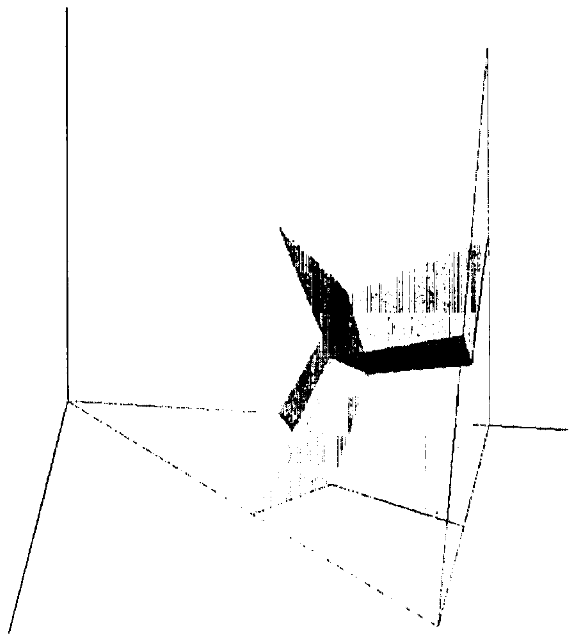
$$\Delta U_k{}^{n+1} - s_5 \Delta t \frac{\partial \Delta U_k{}^{n+1}}{\partial t} - s_6 \frac{\Delta t^2}{2} \frac{\partial^2 \Delta U_k{}^{n+1}}{\partial t^2} = \Delta t \frac{\partial U_k{}^{n}}{\partial t} + \frac{\Delta t^2}{2} \frac{\partial^2 U_k{}^{n}}{\partial t^2}$$

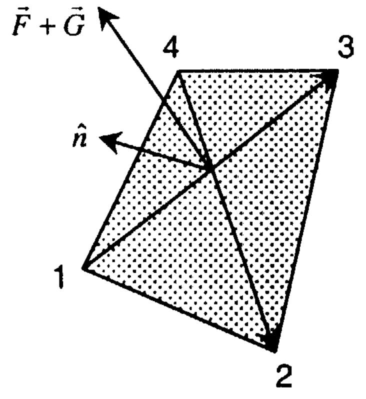
Control Volume Finite Element Method

Vertex Control Volume



Discretization of Tetrahedral Volume

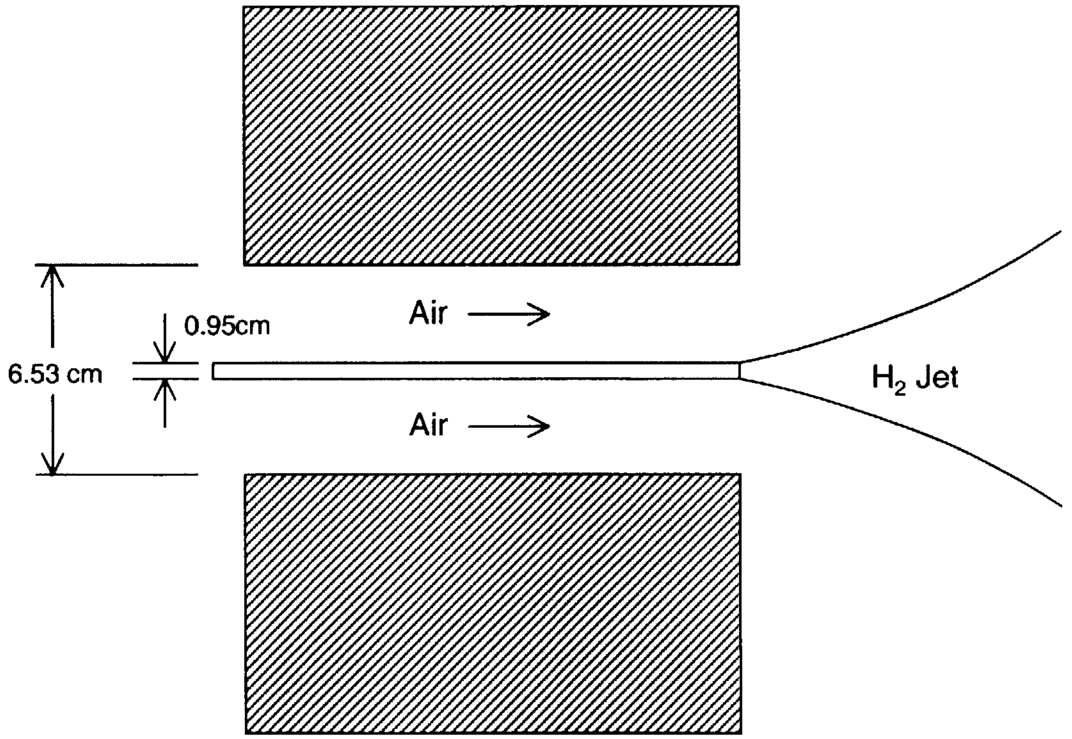




$$\frac{\partial U}{\partial t} = B - \vec{\nabla}\cdot(\vec{F}+\vec{G})$$

$$\int\left[\frac{\partial U}{\partial t} - B\right]dV = -\int \vec{\nabla}\cdot(\vec{F}+\vec{G})dV = -\int (F_i + G_i)n_i d\Gamma$$

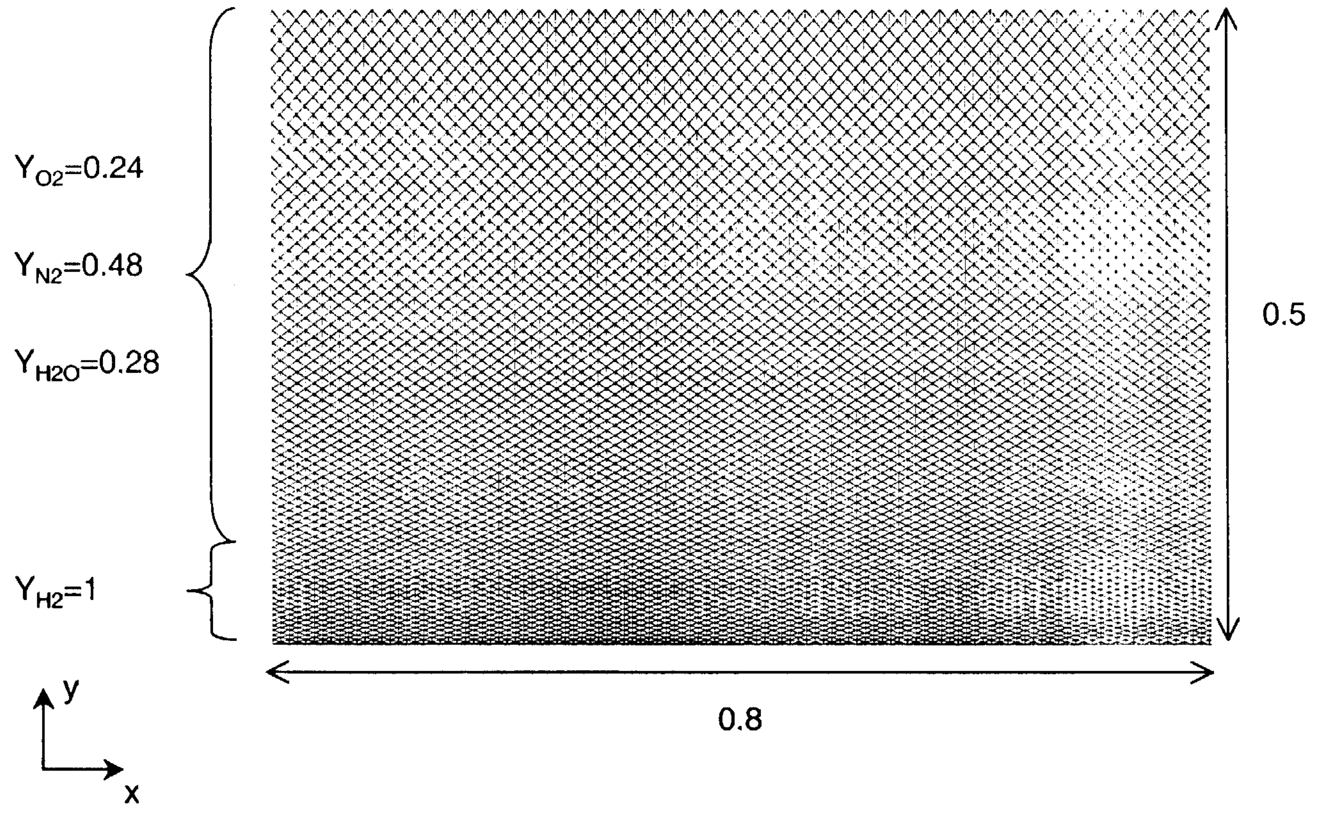
Test Case for Global Hydrogen/Air Combustion Model[1]



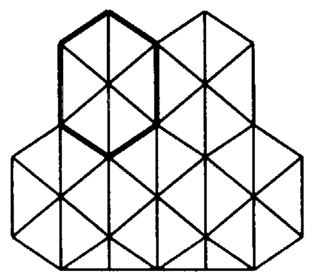
	H2 Jet	Mainstream
Mach	2.0	1.9
Temp (K)	251	1495
Y_{H2}	1.0	0.0
Y_{O2}	0.0	0.24
Y_{N2}	0.0	0.48
Y_{H2O}	0.0	0.28

[1]Rogers, R.C. and Chinitz, W., “Using a Global Hydrogen-Air Combustion Model in Turbulent Reacting Flow Calculations”,AIAA Paper 82-0112, January 1982.

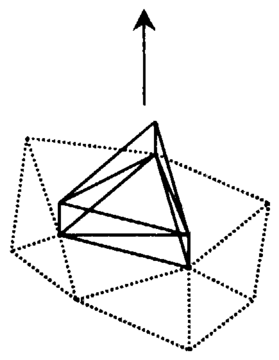
Computational Grid



Hexagonal Elements

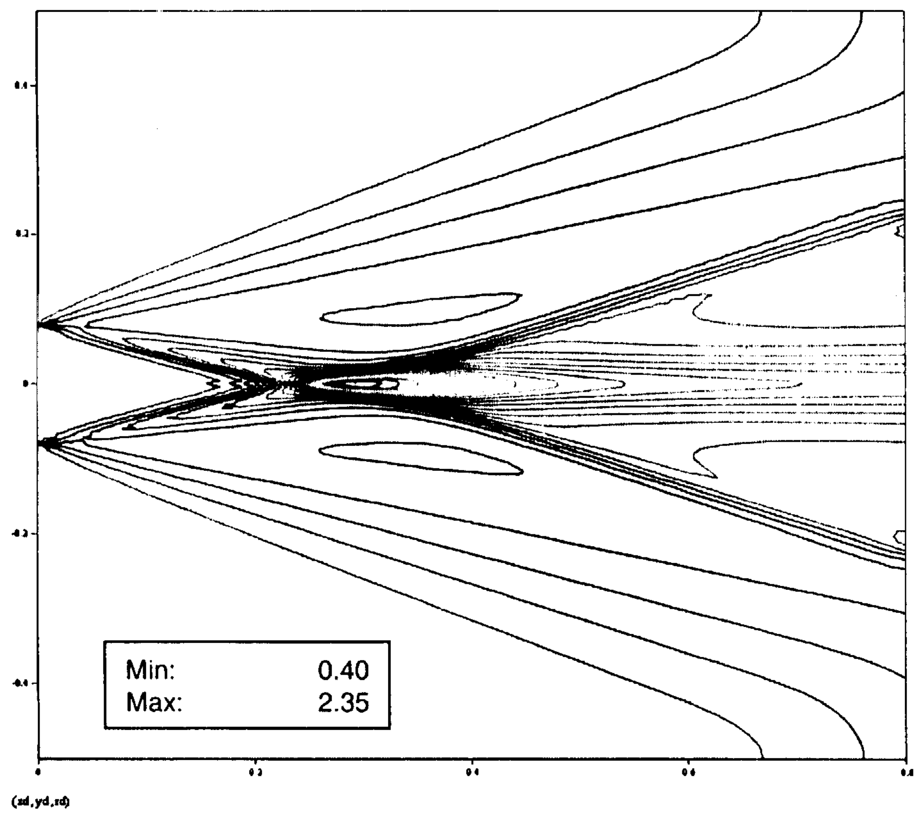


3D Solid Generated by Extruding 2D Planar Surface

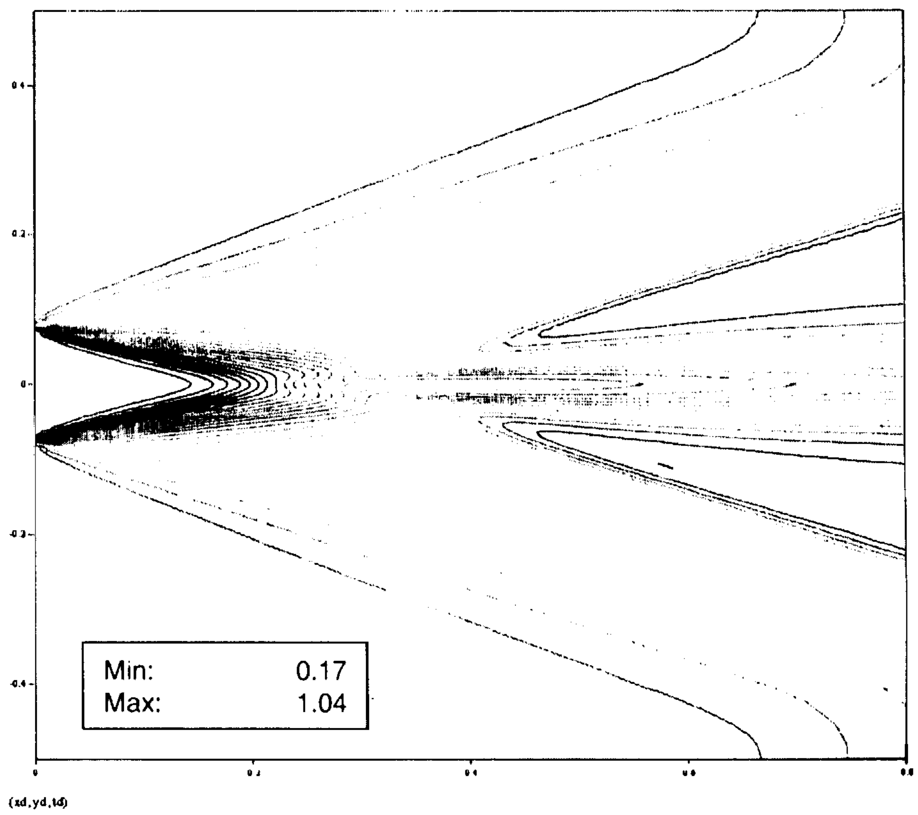


Density and Temperature Contours for Non-reacting Flow-field

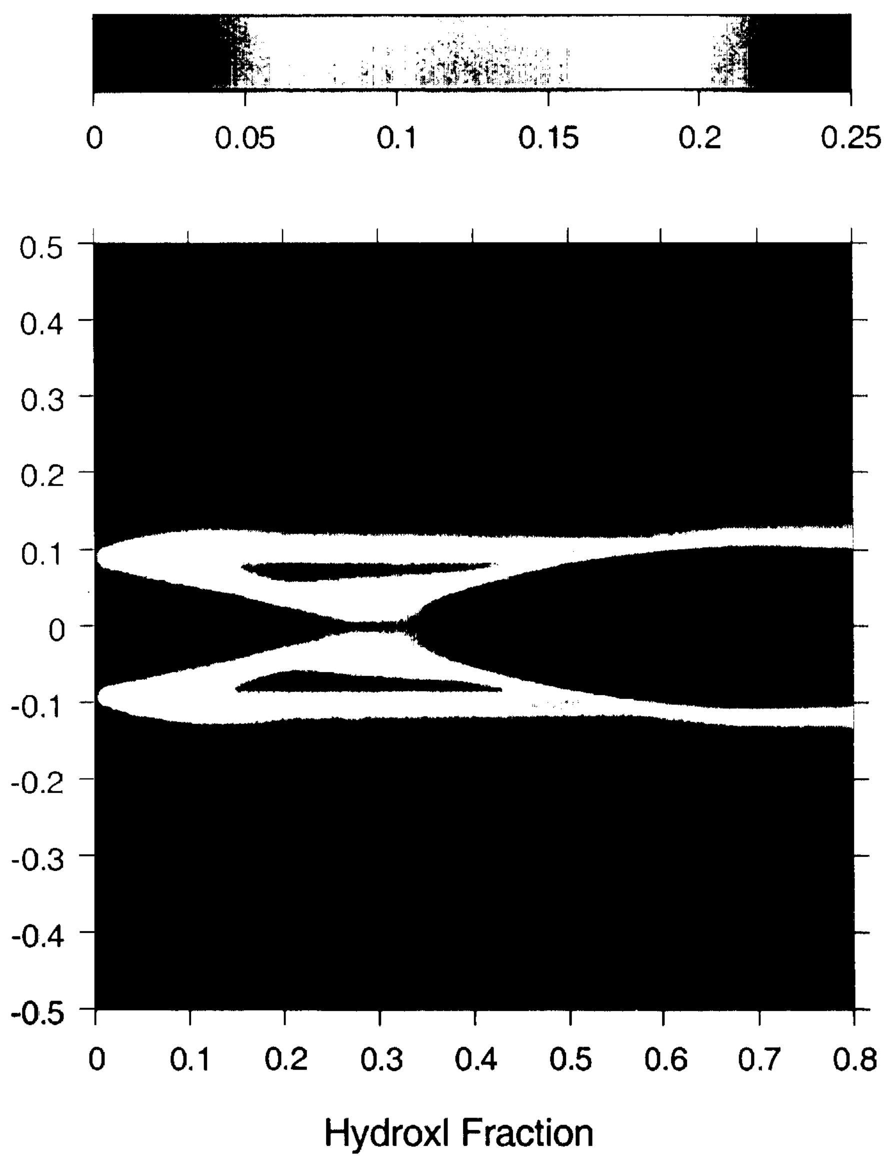
Density

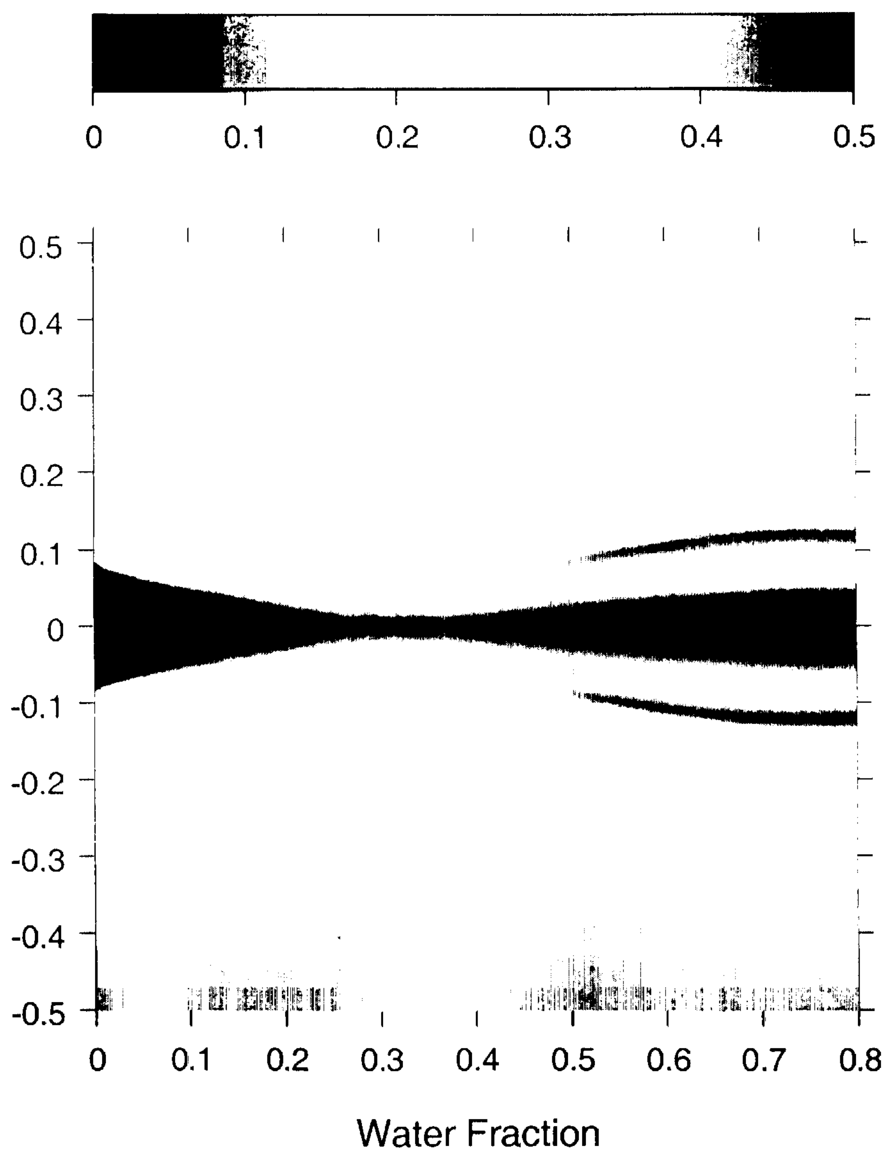


Temperature



Predicted OH/H2O Mass Fractions





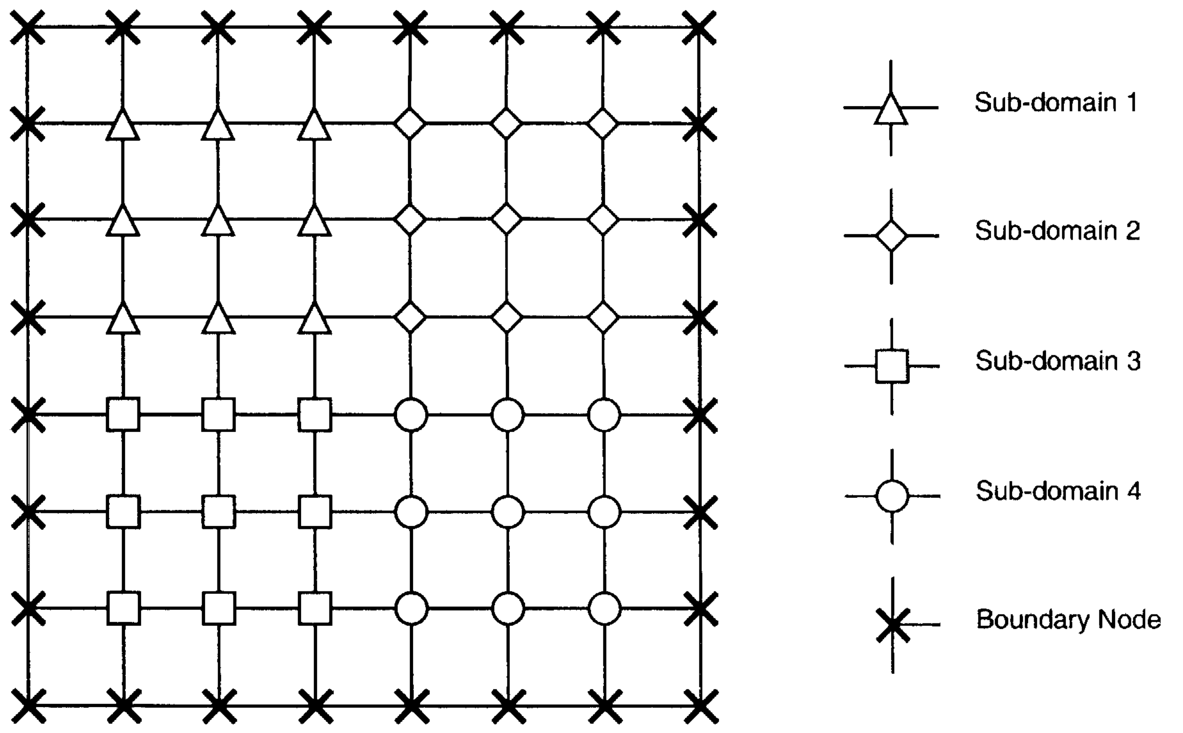
Parallel Programming: Processes and Threads

- When a computational task is delivered to the operating system (OS) of a computer for execution, the OS responds by creating a *process.*
  - The OS allocates memory for the process, provides access to system resources, and schedules time for the process to run.
  - Processes do not normally share resources, but may communicate through mechanisms provided by the OS.
- Within a process, there exists data and program segments and the execution path through the program segment may be thought of as a *thread.*
- If sections of the program may be executed concurrently, then multiple threads through the process may be created.
- Like processes, threads are scheduled for execution by the OS, but share global memory (within the same process) alleviating the need for process level communication.

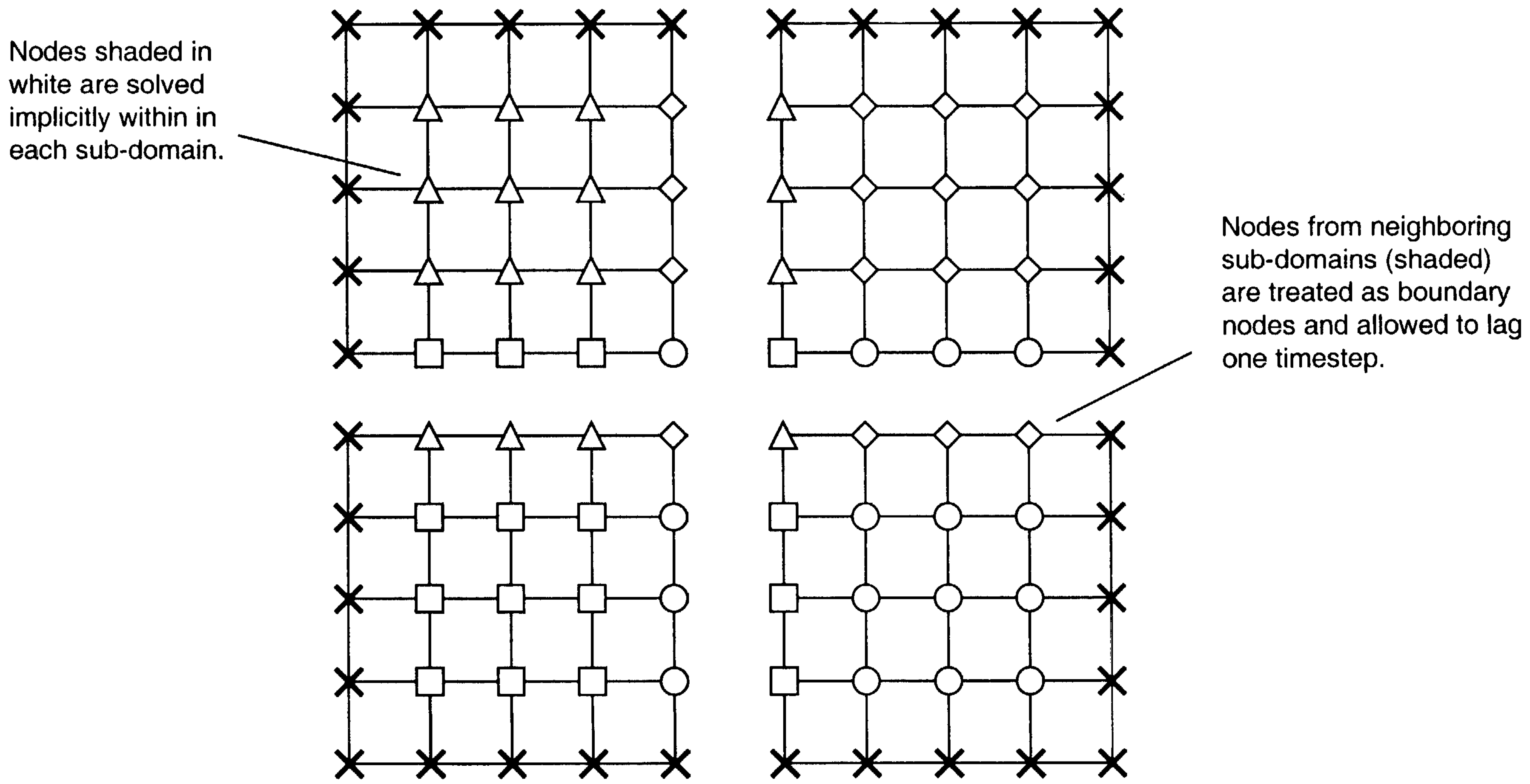
Multi-threaded Programming

- On shared memory multi-processor machines, creating multiple threads is an ideal way to parallelize an application since individual threads may be assigned to separate CPU's by the OS.
- Balancing the computational load between multiple processors is critical to achieving a high degree of parallelism.
- A combined domain decomposition/multi-threaded approach is presented.

Domain Decomposition
Additive Schwarz Method with Overlapping Sub-domains

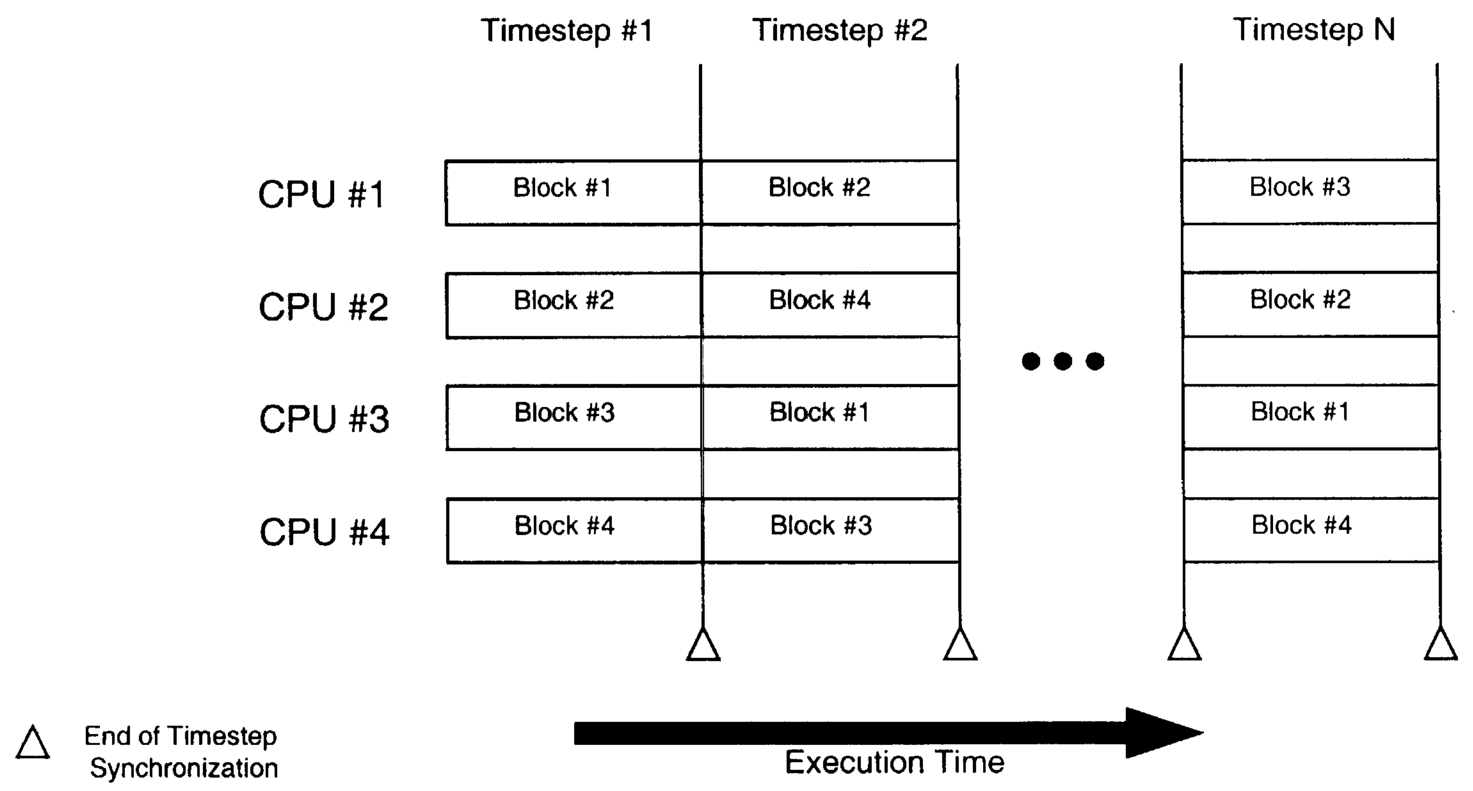


Domain Decomposition
Additive Schwarz Method with Overlapping Sub-domains

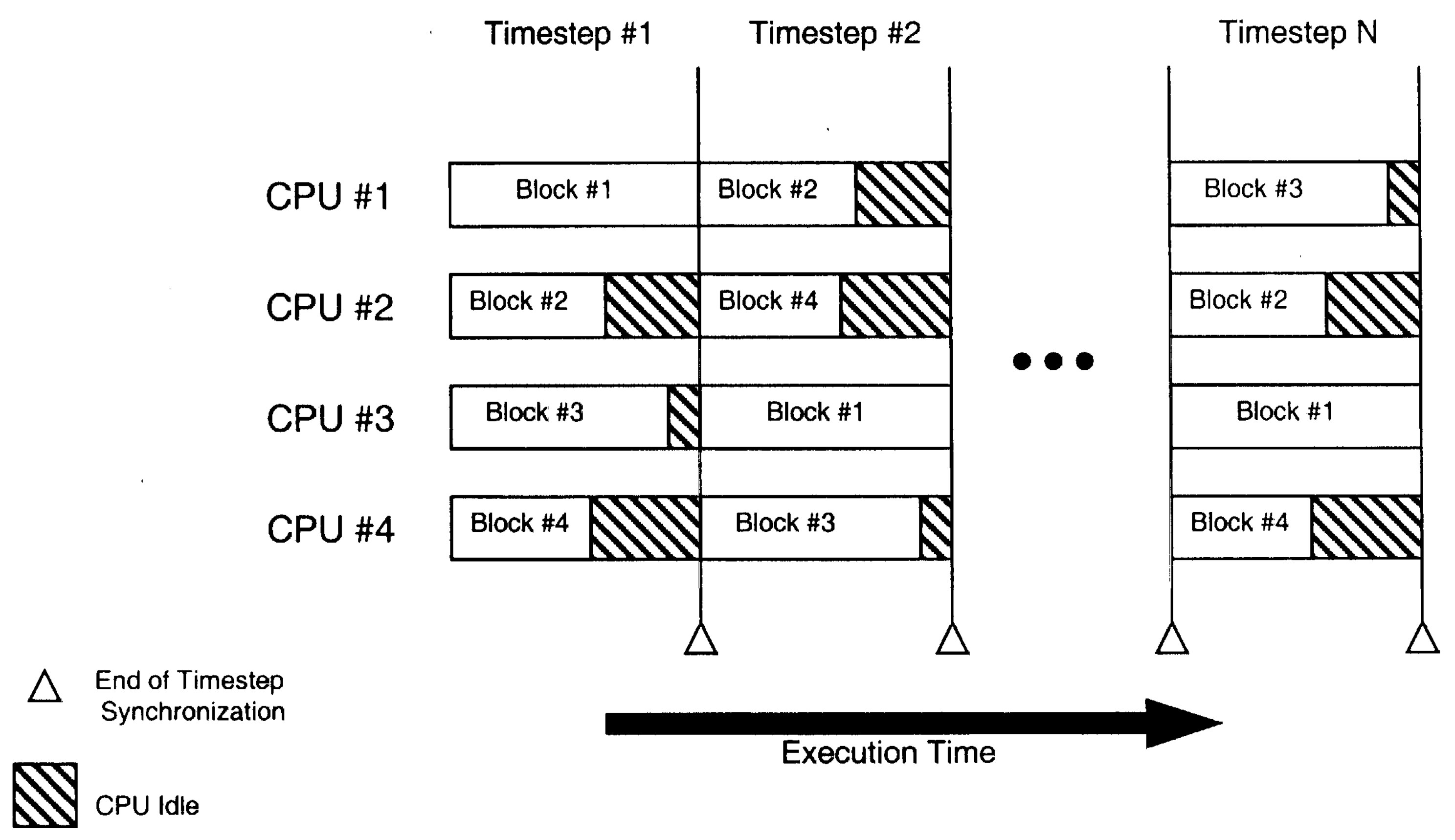

Nodes shaded in white are solved implicitly within in each sub-domain.
Nodes from neighboring sub-domains (shaded) are treated as boundary nodes and allowed to lag one timestep.

*Each interior node is solved in an implicit fashion in exactly one sub-domain.

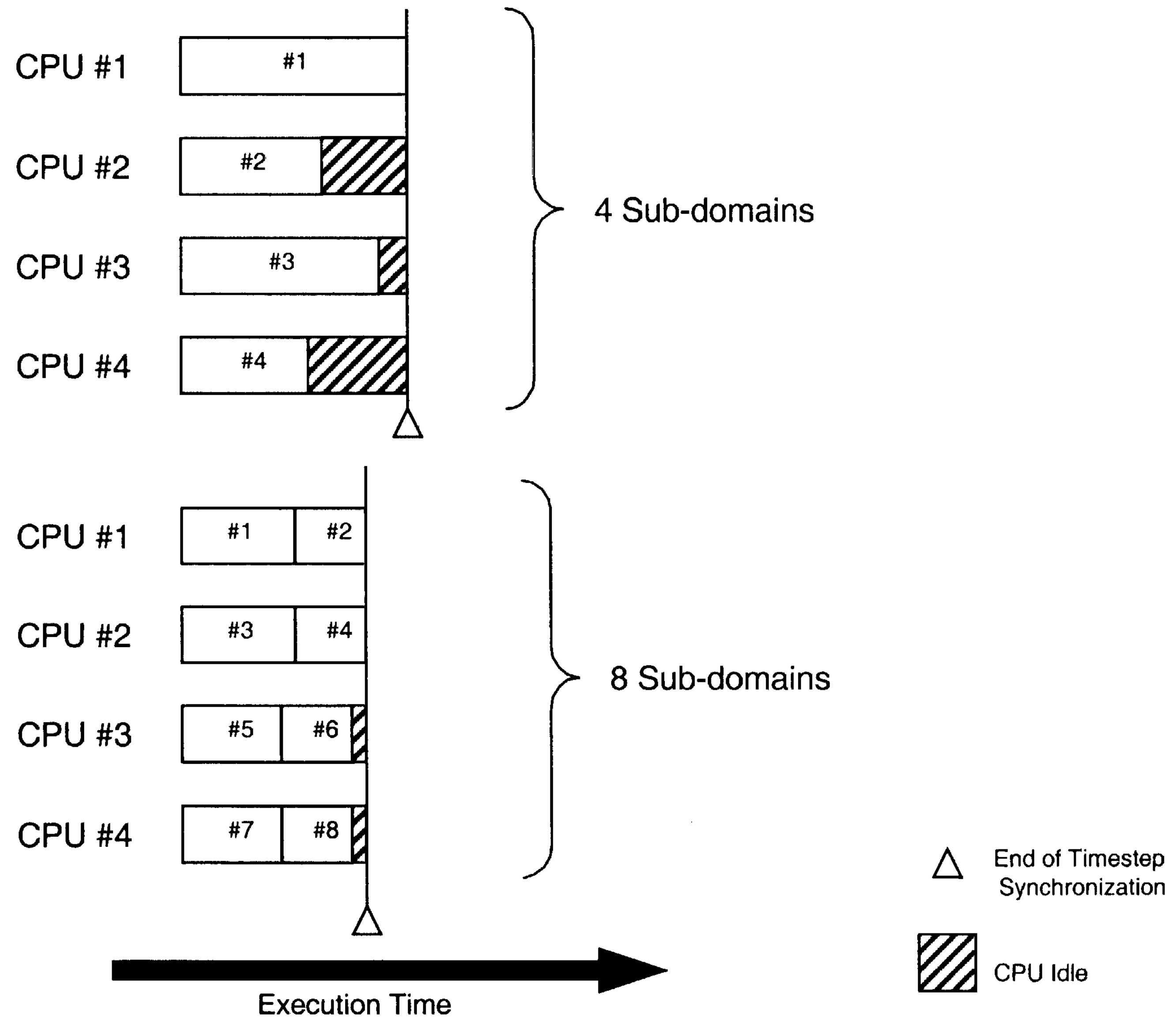
Processor Load Balancing for the Ideal Case



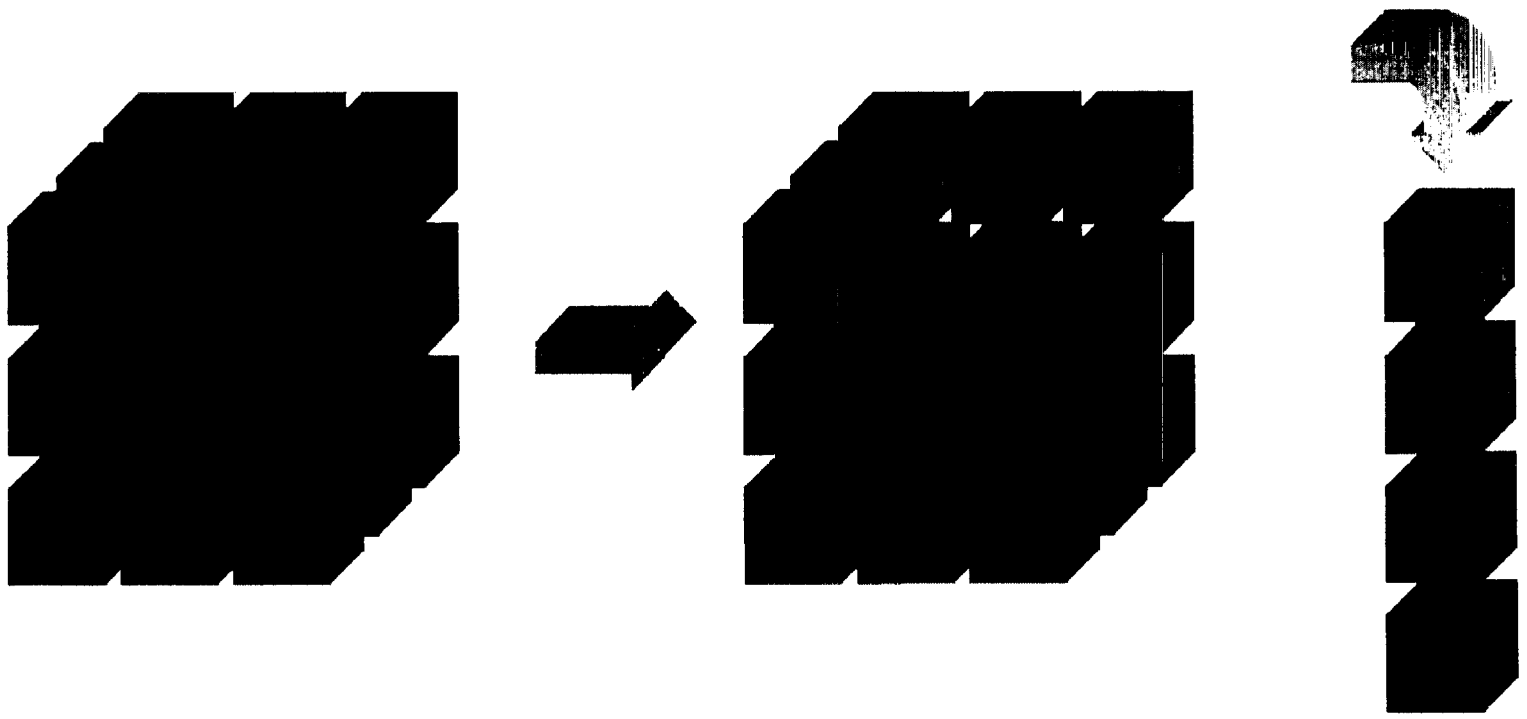
Processor Load Balancing in the "Real World"



Increasing the Number of Sub-domains Improves Load Balancing



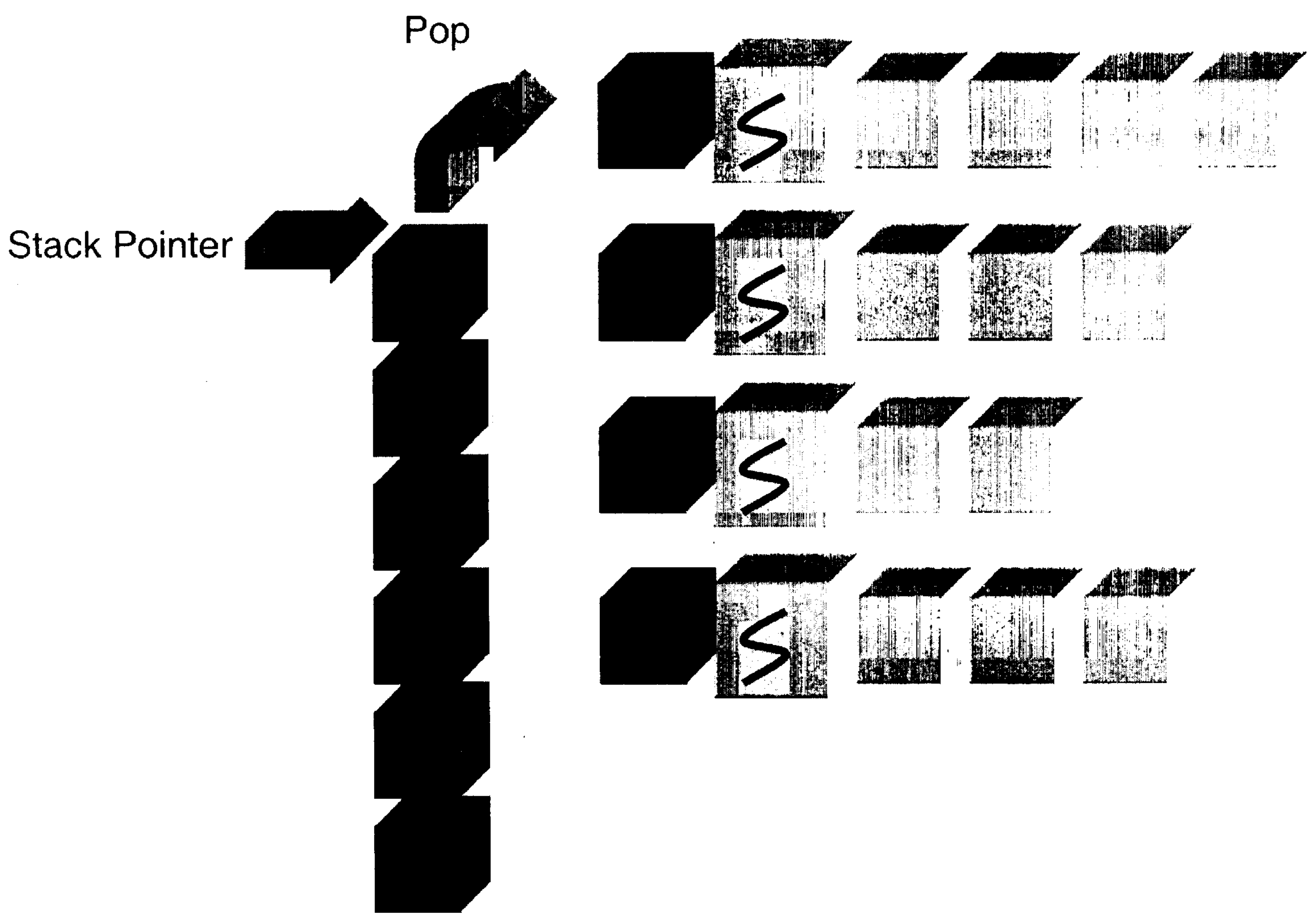
Multi-threaded Programming Implementation



Decompose the domain

Push each sub-domain onto a software stack

Multi-threaded Programming Implementation



Spawn threads and execute until stack is exhausted

Computational Benchmarks

Threads	Grid	Decomposition	CPU Time (hours)	Elapsed Time (hours)	CPU Utilization	Speed-up	Processor	Number of Proc
1	55x41x31	4x4x4	5.05	5.05	100%	1.00	Pentium II	2
2	55x41x31	4x4x4	5.13	2.62	196%	1.93	Pentium II	2
1	55x41x31	4x4x4	4.69	4.72	99%	1.00	Alpha	4
2	55x41x31	4x4x4	5.19	2.66	195%	1.77	Alpha	4
4	55x41x31	4x4x4	5.30	1.42	373%	3.32	Alpha	4
6	55x41x31	4x4x4	5.30	1.40	378%	3.36	Alpha	4
8	55x41x31	4x4x4	5.16	1.37	377%	3.44	Alpha	4

Conclusions and Future Plans

- Preliminary results are encouraging:
  - A more rigorous treatment of the chemical species generation terms may be necessary to relax time-step constraint.
- Incorporate the 8 and 28 reaction H2-Air chemical kinetic mechanisms for comparison to the two step global reaction mechanism.
- Incorporate a Large Eddy Simulation or turbulence model to account for enhanced reaction rates due to turbulent mixing.
- Merge the Navier Stokes and species conservation solvers into one program.
- Migrate the application to a “truly” three dimensional benchmark.