```
\documentstyle[12pt]{article}

% Define own text environment
% Roughly the specifications of scribe article

\textwidth=6.5in
\textheight=8.5in
\oddsidemargin=0.0in
\topmargin=0.0in
\parskip=0.05 in

%\include{macros}       % Include own macros definition file

%\includeonly{propr}

\begin{document}
\bibliographystyle{plain}   % Can have alpha for [He85] style

\pagestyle{empty}           % No page number for title page

%                  Title page
%
\vskip 2 in
\centerline{{\LARGE Parallel Path Consistency}}

\vskip 0.5 in
\centerline{{\large Steven Y. Susswein, Thomas C. Henderson and Joe Zachary}}
\vskip 0.5 in
\begin{center}
Department of Computer Science\\
University of Utah\\
Salt Lake City, UT 84112
\end{center}
\vskip 0.55 in
\centerline{\underline{\bf {\Large Abstract}}}
\vskip 0.15 in

Filtering algorithms are well accepted as a means
of speeding up the solution of the
consistent labeling problem (CLP).  Despite the fact that path
consistency does a better job of filtering than arc consistency, AC is
still the preferred technique because it has a much lower time
complexity.

We are implementing parallel path consistency algorithms on
a multiprocessor and comparing their performance to the best sequential and
parallel arc consistency algorithms.
We also intend to categorize the relation between graph structure and
algorithm performance.
Preliminary work has shown
linear performance increases for parallelized path consistency and also
shown that in many cases performance is significantly better than the
theoretical worst case.  These two results lead us to believe that
parallel path consistency may be a superior filtering technique.  Moreover,
we conjecture that no set of relations exist of $n$ nodes and $m$ labels
which requires more than $mn$ iterations of Path Consistency to make the
relations consistent.

\newpage
%\pagestyle{plain}          % Start page numbering from here
\setcounter{page}{1}        % First page in 1

\section{Introduction}
```

There is a class of problems in computer science known variously as
{\it Consistent Labeling Problems} \cite{Haralick79},
{\it Satisfycing Assignment Problems} \cite{Gaschnig79},
{\it Constraint Satisfaction Problems} \cite{Mackworth77},
etc. We will refer to it as the {\it Consistent Labeling Problem}
(CLP). Many classical computer science problems
such as N-queens, magic squares, and the four color map problem
can be viewed as {\it Consistent Labeling Problems}, along with a number
of
current problems in computer vision.

The basic problem can be looked at abstractly in the form of a graph, in
which we have:
\begin{itemize}
\item A set of {\it nodes}, N = $\{n_1,n_2,\ldots,n_n\}$;
(let ${\mid}N{\mid}=n$).
\item For each node n_i a domain M_i, which is the set of acceptable
labels for that node. Often all the M_i's are the same, giving
$M_1=M_2=\ldots=M_n=M$;
(let ${\mid}M{\mid}=m$).
\item A set of {\it constraint relations} R_{ij}, $i,j=1,n$, which define
the consistent label pairs which can be assigned to nodes n_i and
n_j; i.e., $R_{ij}(l_1,l_2)$ means label l_1 at node n_i with label
l_2 at node n_j is a consistent labeling. Directed arcs give a
visual representation of the relationships.
\end{itemize}
The problem is to find a complete and consistent labeling such that each
node is assigned a label from its label set that satisfies the
constraints induced by all its connected arcs.

For example, take a three node graph where the arcs represent
the relationship ``equals.'' That is, the labels assigned to the
two nodes at the ends of each arc must be equal. If the label
sets for the nodes are:
\begin{itemize}
\item $node_1$: \{1,2,3\}
\item $node_2$: \{2,3,4\}
\item $node_3$: \{3,4,5\}
\end{itemize}
then the only possible solution is:
\newline
$node_1$ = 3
\newline
$node_2$ = 3
\newline
$node_3$ = 3

\subsection {Solutions to CLP}
It can be shown that CLP is NP-complete\cite {Haralick78a}. Thus there are
no known efficient solutions. However, there are a number of ways the problem
can be solved, including {\it generate and test}, {\it standard
backtracking}, {\it Waltz filtering}\cite {Waltz75}, etc.
In standard backtracking, we assign a label to
$node_1$, and using this constraint attempt to find a valid label
for $node_2$. Using these values for nodes one and two, we attempt to
find a valid label for $node_3$, etc. When no valid label
exists for a node, we backtrack and make a new assignment for the last
node. We continue until all nodes have been assigned labels or all
possible assignments have been attempted, and failed.

Mackworth \cite{Mackworth85} has shown that the ``thrashing'' behavior
of standard
backtracking can be reduced by the incorporation of consistency

algorithms ({\it node, arc,} and {\it path} consistency). Mohr and Henderson \cite{Mohr86a} have given an optimal algorithm for arc consistency and an improved algorithm for path consistency.

\begin {itemize}

\item In {\it node consistency}, we look at the label set for a single node and remove any impossible labels.

\item In {\it arc consistency} we look at each pair of nodes and remove those labels which cannot satisfy the arc between them. For example, if we looked at nodes one and two in the above example using arc consistency we would remove the value {1} from $node_1$and the value {4} from $node_2$.

\item In {\it path consistency} we look at groups of three or more nodes (Montanari has shown that if all paths of length two are consistent then the entire graph is consistent, so we actually look at paths of length exactly two).
\end{itemize}

Path consistency does a much better job of filtering than arc consistency, but is also much slower (i.e., requires a lot more computation); as a result, arc consistency is currently the most widely used filtering technique.

\subsection {Parallel Algorithms for AC and PC}

Samal has explored parallel versions of arc consistency \cite{Samal88}. He showed that the worst case performance of any parallel arc consistency algorithm is $O(mn)$.
This means that given a polynomial bound on the number of processors, it takes time proportional to mn to solve the problem in the worst case. Moreover, he explored the dependence of performance on graph structure.

We are interested in providing a similar analysis for parallel path consistency algorithms. We conjecture that the average case time complexity of parallel path consistency is $O(mn)$.
This means that over populations of standard problems and given a polynomial bound on the number of processors, the average time to solve the path consistency problem is proportional to mn. In fact, our preliminary results indicate that the innermost loops of path consistency (i.e., those which update the relations) run in constant time, $O(1)$.
We therefore propose the following conjecture:

\noindent
{\bf Linearity of Parallel Path Consistency}: No set of relations R_{ij}, $i,j=1,n$, exists which requires more than mn iterations of Parallel Path Consistency to make them consistent.

\section{Parallel Path Consistency}

The current best path consistency algorithm (PC-3) has a time complexity of $O(n^{3}m^{3})$, compared to the optimal arc consistency algorithm (AC-4) which has a time complexity of $O(n^{2}m^{2})$\cite{Henderson90a}, but path consistency
does a much better job of pruning the search space. This can be seen by looking at the {\it 4-Queens} problem. Path consistency will prune 50\% of the labels from each node, leaving just two possible positions for each queen; arc consistency on the other hand prunes {\it none} of the labels, leaving the problem at its original complexity.

The main thrust of this research is to define and implement {\it parallel}
versions of the PC algorithms on a multiprocessor to see whether they
can outperform the best AC algorithms when used within search to prune the
search tree at each node.

\subsection{Standalone Parallel PC}
We are currently investigating parallel versions of the PC algorithms
and comparing their performance to each other and to the parallel AC
algorithms measured by Samal. Samal has shown that the best
sequential AC algorithm is not necessarily the best parallel algorithm.
For
each algorithm we will measure its raw speed as well as its speedup linearity,
with the goal of finding a parallel PC algorithm with at least linear
speedup.
Speedup linearity is a measure of how well we are utilizing the additional
processors and is defined as
{\it time on 1 processor$/$(N $\times$ time on N processors)}.

\subsection{Using PC in Search}
The next step involves creating a standard
backtracking program, in which various parallel AC and PC
routines are embedded.
At each node of the search tree we run the chosen AC or
PC code to check for consistency. Again, we are measuring the raw
performance and speedup, as well as the average, minimum, and maximum
search depth and the number of nodes traversed.

\subsection{Finding Worst Case Performance}
Although theoretical worst case performance of sequential PC-1 is of complexity
$O(m^{5}n^{5})$, early experiments have shown actual performance to be much
better (see section 3.3). We are attempting to find and categorize the
worst case performance based on the type of graph and constraint
relation.

N-queens and confused n-queens\cite{Nadel89}
are the standard test cases for performance
measurement and comparison.

\section {Initial Results}

We have conducted some simple experiments. These
experiments support the following claims:
\begin{enumerate}
\item {\it Path consistency} prunes the search space to a greater extent
than {\it arc consistency}.
\item Highly parallelized versions of {\it path consistency} can achieve
near-linear speedup.
\item {\it Path consistency} will normally run in much better than
theoretical worst case performance.
\end{enumerate}

\subsection {Pruning Efficiency of PC vs. AC}
We already had a working version of arc consistency created by Samal,
so PC-1 was coded based on the algorithm
given by Mackworth. Both these programs use identical
system calls to report timing information and were run on a number of
both consistent and inconsistent graphs. These graphs mostly
corresponded to the {\it N-Queens} problem (for various values of {\it
N}), but other graphs were also examined. As expected, arc consistency
ran much faster than path consistency, but path consistency did a
superior job of pruning the search space. As mentioned earlier, a good
example of this is consistent 4-Queens. Figure 1 shows number of nodes
expanded for n-queens ($n=4,6,8,10$).

```
\begin{figure}
\vspace{6in}
\caption{Number of Nodes Expanded in N-Queens for AC and PC}
\end{figure}

\subsection {Parallel PC-1}
As a next step, we modified the PC-1 program mentioned above to run as a
parallel program on the Butterfly.  We employed a straightforward
parallelization, where the number of parallel processes generated is
based on the size of the initial graph.  Larger graphs have shown an
approximately linear speedup, up to the number of processors available
(see Table~1).  Note that the number of iterations varies slightly due to
interactions caused by the parallelization, and the speedup remains linear
only for equal iteration counts.
\begin{table}
\begin {center}
\begin{tabular}{|c|r|c|c|c|} \hline
 & & & \multicolumn{2}{c|}{\em speedup linearity} \\
{\em processors} & {\em raw time (ms)} & {\em iterations} & {\em 2 iterations}
        & {\em 3 iterations} \\ \hline
1s & 602980 & 2 & 1.00 & \\
1p & 626305 & 2 & 0.96 & \\
2 & 322272 & 2 & 0.94 & \\
3 & 213479 & 2 & 0.94 & \\
4 & 244888 & 3 &      & 0.62 \\
5 & 128830 & 2 & 0.94 & \\
6 & 169108 & 3 &      & 0.59 \\
7 & 142597 & 3 &      & 0.60 \\
8 & 123289 & 3 &      & 0.61 \\
9 & 113087 & 3 &      & 0.59 \\
10 & 101053 & 3 &     & 0.60 \\
11 & 93206 & 3 &      & 0.59 \\
12 & 84602 & 3 &      & 0.59 \\
13 & 78071 & 3 &      & 0.59 \\
14 & 72184 & 3 &      & 0.60 \\
15 & 44201 & 2 & 0.91 & \\ \hline
\end{tabular}
\end{center}
\begin {center}
Note: 1s is sequential code and 1p is parallel code
\end{center}
\caption{Speedup Linearity for 16-Queens using PC-1}
\end{table}

\subsection {Worst Case Performance}
The graph input to PC-1 is encoded in the form of an $nm *
nm$ binary matrix.  The algorithm iterates over this matrix until
two successive iterations yield no change in the matrix.  Each
iteration is of complexity $O(m^3n^3)$ and can only simplify the matrix
(i.e., change a ``1'' to a ``0'').  Since each iteration simplifies at least
one element in the matrix, we require as a worst case $m^2n^2$
iterations, yielding a worst case performance of $O(m^5n^5)$.

Since the input matrix defines both the list of possible labels for each
node {\it and} the constraint relation between nodes, it is possible to
exhaustively examine all possible relation constraints
for small values of {\it m} and {\it n} through a brute-force approach
of constructing all possible input matrixes.  The purpose of this experiment
was to find which constraint relations produced the worst results
(greatest number of iterations).
While we haven't been able to fully characterize which constraint relations
produced the most iterations,
we were surprised by the maximum and average number of iterations
```

required.
Using values of $m=2$ and $n=3$
yielded a worst case performance of 5 iterations (compared to a
theoretical worst case of 30 iterations) and an
average case performance of 2.07 iterations (see Figure 2).
\begin{figure}
\vspace{6in}
\caption{Avg. and Max. Iterations for all relations of $m=3$ and $n=2$.}
\end{figure}

Additional experiments varying the value of {\it m} and {\it n} for a
fixed relation showed that the number of iterations required remains
small and relatively constant for at least some relations. If found to
be generally true for all relations this would make parallel path
consistency even more attractive. Each iteration in PC-1 can be highly
parallelized, but the iterations themselves are performed in
sequence. The number of iterations
required (whose upper bound is theoretically m^2n^2) places an
upper bound on the
efficiency of parallel PC; if the number of iterations required is
found to be small
and relatively constant for large values of {\it m} and {\it n}, then
parallel path consistency may prove to be a superior filtering
technique.

\section {Tools and Facilities}

All the code is being written in standard {\it C}. Timing
information is gathered using standard {\it Unix} system calls (for
the sequential code) and {\it Uniform} built-in timing routines (for
the parallel code).

\subsection {DECStation 3100}
Sequential code is developed and run on a dedicated DECStation
3100, a high-performance RISC workstation. Code developed here under
{\it ULTRIX} is source-code compatible with the University Bobcat
workstations, but its high performance (approximately $3\times$ an HP370) and
lack of contending jobs means that large runs can be completed quickly.

\subsection {Butterfly GP1000}
Parallel code is being developed and run on the BBN
Butterfly multiprocessor. The Butterfly offers two means of accessing
its multiprocessor features: direct system calls to the {\it Mach}
operating system, and the {\it Uniform} system. The {\it Uniform}
system consists of a library of routines which allow easy access to the
multiprocessing features. While not as powerful as direct {\it
Mach} calls, it is much easier to use and supplies all the
features needed to implement parallel path consistency.

The Butterfly is configured with eighteen nodes, which
will be sufficient for development and testing and to show the effect of
parallelized PC, but we also hope to gain access to a 40 node Butterfly
located at Cornell University to verify that my results hold for a
larger degree of parallelization.

\subsection {Connection Machine}
In future work, we plan to represent and compute
Path Consistency as an outer product
\cite{Marsden90} on a
fine grain connection machine at Los Alamos. We hope to determine from
this whether the speedup is sufficient to motivate the investigation of a
special-purpose integrated circuit.

```
% Include bibliography files here
\bibliography{general,bibliography}

% Table of Contents
%\vfill\eject
%\pagenumbering{roman}
%\setcounter{page}{1}
%\tableofcontents

\end{document}
```