Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 249–261. http://www.scpe.org
ISSN 1895-1767

A TOOL FOR A TWO-LEVEL DYNAMIC LOAD BALANCING STRATEGY IN SCIENTIFIC APPLICATIONS*

RICOLINDO L. CARIÑO† AND IOANA BANICESCU‡

Abstract. This paper describes a dynamic load balancing tool intended for computational investigators who have little familiarity with programming for a message-passing environment. Motivated by the PAR DOALL directive available in some compilers for shared-memory systems, the tool is designed to simplify the manual conversion of sequential programs containing computationally intensive one- or two-dimensional loops with independent iterates into parallel programs that execute with high efficiency on general-purpose clusters. The tool implements a dynamic loop scheduling strategy to address load imbalance which may be induced by the non-uniformity of loop iterate times, and by the heterogeneity of processors. The tool is based on the Message Passing Interface library for wide availability. Experimental results of two scientific applications that utilize the tool on a Linux cluster are presented to demonstrate sample achievable performance, and to underscore the effectiveness of the two-level dynamic load balancing strategy.

Key words. dynamic load balancing, loop scheduling, image denoising, vector functional coefficient autoregressive model

1. Introduction. Automatic parallelization of sequential programs containing computationally-intensive parallel loops is supported by many compilers for shared-memory environments. For example, on a Sun multiprocessor and with a Sun Fortran 77/90 compiler [38], simple insertion of the `C$PAR DOALL` directive before a parallel loop will instruct the compiler to generate parallel code. Similar directives are also provided in Sun C [37] and OpenMP [9]. A clause attached to the directive may specify Guided self scheduling [42] or factoring [26] as the loop scheduling technique for load balancing, or a fixed chunk size may be supplied. Other scheduling techniques have been developed, such as trapezoid self scheduling [46], weighted factoring [25], adaptive weighted factoring [5, 7, 11, 12], and, adaptive factoring [4, 6]; however, at the time writing, these techniques are not supported by the said clause.

In contrast, much manual effort is required to convert a serial program with a parallel loop so that it will execute on a message-passing platform. Code for explicitly assigning the loop iterates to processors have to be written, which can be a daunting task for those not familiar with developing message-passing programs. The code can statically assign iterates to the participating processors before runtime—a simple strategy that is appropriate if the iterates have uniform execution times and if the processors are homogeneous and start at the same time. Otherwise, chunks of iterates have to be dynamically assigned during runtime to processors that become idle. In this case, the chunk sizes must be chosen judiciously such that the processors will all finish at the earliest possible time. The loop scheduling techniques mentioned above may be utilized for computing the chunk sizes.

The contribution of this paper is a tool that is an "analogue" of the `C$PAR DOALL` directive for parallel loops in applications that will execute on message-passing clusters. The tool is designed to simplify the conversion of a sequential program into a message-passing parallel version with dynamic load balancing, but without extensive code modification. Essentially, one statement initializes the tool and five statements convert a parallel loop, leaving the code of loop body untouched. High performance is expected from the resulting parallel version because of the added dynamic load balancing capability to address both algorithmic and systemic sources of load imbalance. The target users of the tool are scientists who write code for novel computational techniques in a serial language, but require a cluster to extensively investigate the properties of the technique and to exercise the code. Examples of target users are statisticians who are developing new estimation procedures for which statistical properties require study through Monte Carlo simulations, and numerical analysts conducting accuracy studies or parametric studies for new numerical methods.

The rest of this paper is organized as follows. Section 2 identifies the class of applications which can take advantage of the proposed load balancing tool. The section also reviews dynamic loop scheduling techniques that have been utilized for determining processor loads. Section 3 describes the design of the tool and its

*This work was partially supported by the following National Science Foundation grants: ITR/DMS 0313274, EPS 0132618 and, CAREER 9984465.

†Center for Advanced Vehicular Systems – HPCC, PO Box 5405, Mississippi State University, MS 39762, rlc@cavs.msstate.edu

‡Department of Computer Science and Engineering, PO Box 9637, and Center for Computational Sciences – HPCC, PO Box 9627, Mississippi State University, MS 39762 ioana@cse.msstate.edu

interface to applications. Section 4 illustrates the use of the tool in two real applications: the simulation of a hybrid model for image denoising, and the investigation of the statistical properties of estimators and hypothesis tests related to the vector functional coefficient autoregressive (VFCAR) model for multivariate nonlinear time series. Section 5 gives a summary and describes future work.

2. Parallel loops and dynamic loop scheduling. The proposed dynamic load balancing tool is intended for serial applications that contain straightforward parallel loops (1D-loops) or nested parallel loops (2D-loops), as illustrated in Figure 2.1, using Fortran 90 notation.

```
a) 1D-loop
  ...
  DO I=1,N
    ...  I-ITERATE
  END DO
  ...
```

```
b) 2D-loop
    ...
  DO J=1,M  ! J-LOOP
     ...  PART A OF J-ITERATE
    DO I=1,N(J)  ! I-LOOP
     ...  I-ITERATE of J-ITERATE
    END DO ! i
    ...  PART B OF J-ITERATE
  END DO ! j
```

FIG. 2.1. *Target loops*

Parallel loops, or loops with no dependencies among their iterates, are major sources of concurrency in scientific applications. These are frequently targeted for parallelization to reduce application execution time, and therefore, to achieve high performance. The iterates of a parallel loop can be executed in any order or even simultaneously without affecting the correctness of the computations. However, minimizing the loop completion time is not straightforward. Factors such as the nonuniformity of iterate execution times, heterogeneity of the processors, and operating system interference during loop execution, give rise to unbalanced processor workloads, which ultimately lead to application performance degradation. These factors, arising from algorithmic and systemic irregularities that may not be known before the application starts, require the integration of dynamic load balancing into the strategy for parallel loop execution.

Load balancing during the execution of a parallel loop is achieved through *dynamic loop scheduling*, where chunks of loop iterates are executed concurrently by the participating processors. Many techniques for computing the chunk sizes have been proposed and implemented, including non-adaptive [32, 42, 26, 46, 25] and adaptive [5, 7, 4, 6, 11, 12] techniques, in addition to simple static chunking and self scheduling. The simple techniques and some of the non-adaptive techniques [42, 26] have been incorporated into compiler technologies for shared-memory environments, such as multiprocessors from Sun Microsystems [37, 38]. Thus, automatic parallelization could be achieved through judicious insertion of a compiler directive immediately before a parallel loop. To the authors' knowledge, this facility is not yet available for message-passing clusters, which typically have far more processors and interesting sources of irregularities than shared-memory multiprocessors.

A strategy for executing 2D-loops on shared-memory multiprocessors has been proposed and theoretically investigated [10, 24, 45]. This strategy describes a feedback-guided self scheduling (FGDLS) algorithm which uses a feedback mechanism to schedule a parallel loop within a sequentially executed outer loop. It has been shown to perform well for scheduling problems for which the load associated with the parallel loop changes relatively slowly as the outer loop executes sequentially. Sufficient conditions have been established for the convergence of the algorithm. The FGDLS differs significantly from the strategy underlying the tool proposed in this paper; here, the iterates of the outer loop are executed in parallel and the target environment is a message-passing architecture. Thus, at least two additional levels of complexity are addressed by the tool.

Some load balancing libraries have been developed for clusters [27, 34, 21], but these are for specific classes of applications that utilize standardized data structures. Previously, the authors have designed a load balancing tool which requires the user to provide a routine that encapsulates the computations of a loop iterate, and routines for migrating data between processors [13].

3. Tool design. A user of the tool proposed in this paper may be someone who has developed a new computational technique and has written a sequential program to investigate the properties of the technique or to exercise the code implementing the technique. The sequential program is essentially a computationally-

intensive parallel loop, and the user has realized that he will be able to conduct more extensive experiments and to publish results sooner if his investigations are carried out on a parallel system. Although automatic parallelization is supported by many compilers for shared-memory environments, the user prefers a message-passing cluster because of platform availability or because of the necessity for more processors. This is the ideal scenario in which the proposed dynamic load balancing can be utilized.

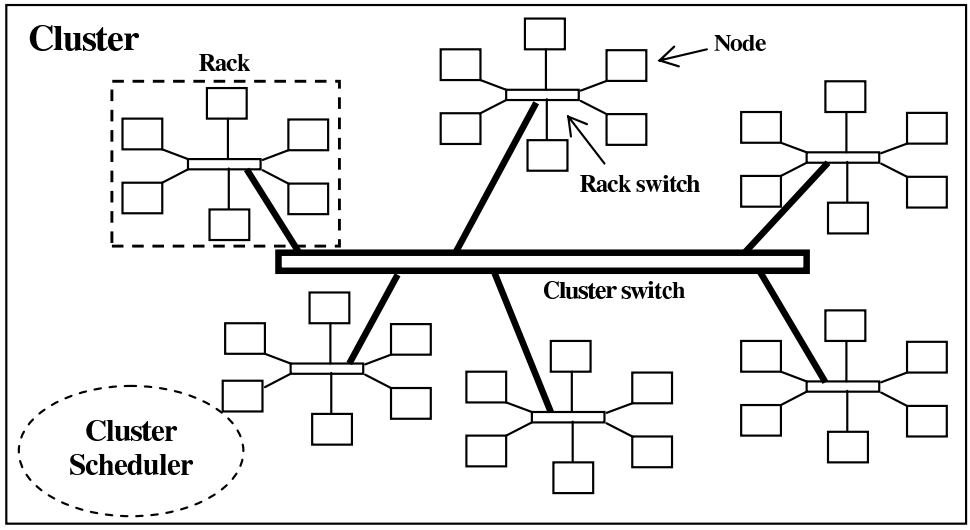

FIG. 3.1. *A popular interconnection network for clusters*

The tool simplifies the parallelization and load balancing of applications that contain computationally-intensive 1D- and 2D-loops (Figure 2.1) on message-passing clusters. These clusters are usually organized into racks that are connected by a cluster switch, each rack consisting of a number of nodes connected by a rack switch, and each node containing one or more processors. Figure 3.1 illustrates a popular interconnection configuration. Heterogeneity is inherent in such a cluster, more so if it was constructed incrementally over a period of time, because the processors would have different capabilities. Rates of communication between processors are also variable. Typically, the cluster scheduler attempts to assign nodes from a single rack to a job for efficient communications. Even with excellent job scheduling algorithms, the scattering of processors across a number of racks occurs with a high probability, especially for jobs that request large numbers of processors. Thus, applications running on clusters typically need to incorporate load balancing for highest possible performance.

Figure 3.2 illustrates the modification of the code of the 1D-loop to integrate the tool. Only a few lines are added: in summary, the original `DO` loop (capitalized) is converted to a `while` loop where chunks of iterates can be executed concurrently on different processors. Since the `I-ITERATE` invokes CPU-intensive computations, which may be expressed in hundreds or thousands of lines of code, the additional code to integrate the tool constitutes a tiny percentage of the total number of lines of code for the application.

The module `DLS` (abbreviation for Dynamic Loop Scheduling) contains the type definition of `infoDLS` and the codes for the `DLS_*` routines. Based on the Message-Passing Interface (MPI) library [22], the routines implement a scheduler–worker strategy of load balancing, where the scheduler participates in executing loads, in addition to being responsible for assigning loads.

The `DLS` routines used in Figure 3.2 are:

`DLS_Setup(MPI_COMM_WORLD,info)` initializes a dynamic loop scheduling environment (Figure 3.3) on `MPI_COMM_WORLD`. Information about this environment is maintained in the data structure `info`.

`DLS_StartLoop(info,1,N,method)` is the synchronization point for the start of loop execution. `(1,N)` is the loop range, and `method` is a user-specified index for the selected loop scheduling technique. The following techniques are implemented: static scheduling, a modification of fixed size chunking [32], guided self scheduling [42], factoring [26], variants of adaptive weighted factoring [5, 7, 11, 12], and adaptive factoring [4, 6].

`DLS_Terminated(info)` returns true if all loop iterates have been executed.

```
program Application_with_1D_loop
  use DLS
  include 'mpif.h'
  type (infoDLS) info
  integer method, iStart, iSize, iIters
  double precision iTime
  integer mpierr
  call MPI_Init (mpierr)
  ...
  method = ...
  call DLS_Setup (MPI_COMM_WORLD, info)
  call DLS_StartLoop (info, 1, N, method)
  do while ( .not. DLS_Terminated(info) )
    call DLS_StartChunk(info,iStart,iSize)
    DO I=iStart, iStart+iSize-1
      ...  I-ITERATE
    END DO
    call DLS_EndChunk (info)
  end do
  call DLS_EndLoop (info, iIters, iTime)
  ...
```

FIG. 3.2. *Dynamic load balancing of an application containing a 1D-loop.*

`DLS_StartChunk(info,iStart,iSize)` returns a range for a chunk of iterates. This range starts with iterate `iStart` and contains `iSize` iterates.
`DLS_EndChunk(info)` signals the end of execution of a chunk of iterates. Internally, a worker processor requests its next chunk from the scheduler.
`DLS_EndLoop(info,iIters,iTime)` is the synchronization point at the end loop execution. `iIters` is the number of iterates done by the calling processor, and `iTime` is the cost (in seconds) measured using MPI_Wtime(). `iIters` and `iTime` are useful for assessing the performance gains achieved by dynamic load balancing. For example, the sum of the `iTime`s from all participating processors gives an estimate of the cost of executing the loop on a single processor.

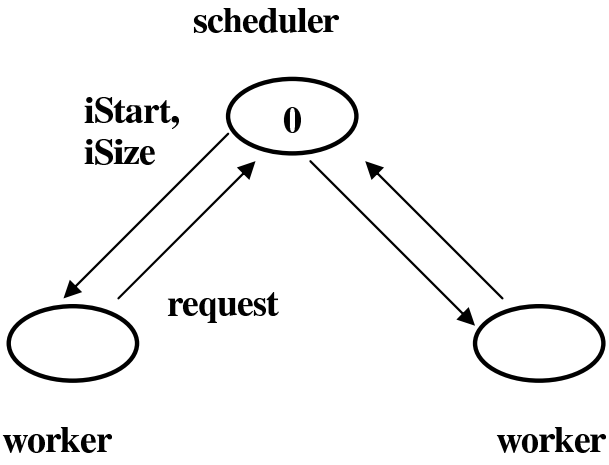

FIG. 3.3. *Scheduler-worker strategy for dynamic loop scheduling*

After loop execution, the results of the computations (in `I-ITERATE`) will be distributed among the participating processors. A reduction operation like MPI_Reduce() may be necessary to collect the results in one processor, or MPI_Allreduce to make the results available to all processors in `MPI_COMM_WORLD`. This would be the responsibility of the user, since DLS only manipulates the indices of the loop. Information about the mapping of the chunks of iterates to processors is maintained in the `chunkMap` component of the `infoDLS` structure.

Figure 3.4 illustrates the integration of DLS into an application containing a 2D-loop. (The original code is capitalized.) An iterate of the outer loop may be composed of a series of `I-LOOP`s; the code for each one goes through the same modification.

```
program Application_with_2D_loop
  use DLS
  include 'mpif.h'
  ...
  type (infoDLS) iInfo, jInfo
  integer iMethod, iStart, iSize
  integer jMethod, jStart, jSize
  integer iIters, jIters
  double precision iTime, jTime
  integer coordinator, minSize, maxSize
  ...
  iMethod = (loop scheduling technique)
  jMethod = (loop scheduling technique)
  minSize = (min. processors for inner loop)
  maxSize = (max. processors for inner loop)
  coordinator = 0
  call DLS_GroupSetup (MPI_COMM_WORLD,&
    coordinator,minSize,maxSize,jInfo,iInfo)
  call DLS_StartLoop (jInfo, 1, M, jMethod)
  do while ( .not. DLS_Terminated(jInfo) )
    call DLS_StartChunk (jInfo, jStart, jSize)
    DO J=jStart, jStart+jSjze-1  ! J-LOOP
      ...  PART A OF J-ITERATE
      call DLS_StartLoop (iInfo,1,N(J),iMethod)
      do while ( .not. DLS_Terminated(iInfo) )
        call DLS_StartChunk(iInfo,iStart,iSize)
        DO I=iStart, iStart+iSize-1  ! I-LOOP
          ...  I-ITERATE of J-ITERATE
        END DO ! i
        call DLS_EndChunk (iInfo)
      end do ! while
      call DLS_EndLoop (iInfo, iIters, iTime)
      ...  PART B OF J-ITERATE
    END DO ! j
    call DLS_EndChunk (jInfo)
  end do ! while
  call DLS_EndLoop (jInfo, jIters, jTime)
  ...
```

FIG. 3.4. *Parallelization and load balancing of a 2D-loop.*

The environment to execute a 2D-loop is initialized by `DLS_GroupSetup()`, which splits `MPI_COMM_WORLD` into a number of non-overlapping communicators `iComm`s and one communicator `jComm` (see Figure 3.5). The processors residing in the same rack are combined into an `iComm` for efficient communications, subject to the constraint that the size of `iComm` is in the range [`minSize, maxSize`], to match the amount of concurrency in the inner loop. Processors from different racks may be combined in an `iComm` to satisfy `minSize`, or processors from a single rack may be split into many `iComm`s to satisfy `maxSize`. The `jComm` is comprised of the `coordinator` and the rank-0 processors of the `iComm`s. Information regarding this two-level setup is stored in the data structures `jInfo` and `iInfo`. Except for the `coordinator`, all processors participate in executing iterates. The iterates of the `J-LOOP` are dynamically scheduled in `jComm`, following the scheduler-worker strategy depicted by Figure 3.3. Chunks of `J-ITERATE`s are concurrent executed on the `iComm`s, each `iComm` also following the same scheduler-

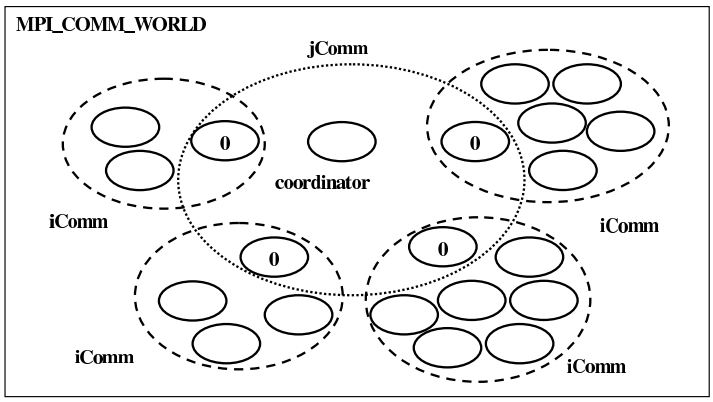

FIG. 3.5. *Two-level setup for 2D-loops. Dynamic load balancing occurs simultaneously on all the communicators.*

worker strategy as the `jComm`. In fact, the same `DLS` routines are used, but with different arguments, notably, `iInfo` in an `iComm` and `jInfo` in the `jComm`.

Within an `iComm`, the processors execute a chunk of `J-ITERATE`s sequentially, in lock step. The processors redundantly compute `PART A` and `PART B` of a `J-ITERATE`. However, the processors cooperatively execute the `N(J)` iterates the `I-LOOP`. For this strategy to be applicable, `M` must be significantly larger than the number of `iComm`s, and that each `N(J)` must be significantly larger than the number of processors in an `iComm`. Otherwise, the number of chunks to be scheduled may not be sufficient in order to achieve good load balancing.

4. Applications. The dynamic load balancing tool has been integrated into a number of scientific applications. To illustrate the utility and the performance improvement achievable by the tool, this section presents timing results for the simulation of an image denoising model, and the simulation of a vector functional coefficient autoregressive (VFCAR) model for multivariate nonlinear time series.

These applications were executed on the heterogeneous general-purpose EMPIRE cluster of the Mississippi State University. The cluster can be abstracted as in Figure 3.1 and has a total of 1038 processors. A rack contains 32 nodes of dual 1.0GHz or 1.266GHz Pentium III processors and 1.25GB RAM. Each node is connected to a 100Mb/s Ethernet rack switch. The rack switches are connected by a gigabit Ethernet cluster switch. Installed software includes RedHat Linux and PBS. The general submission queue allows 64-processor, 48-hour jobs; a special queue allows 128-processor, 96-hour jobs from a restricted set of users. According to the Top 500 Supercomputer Sites list published in June 2002, EMPIRE then was the 126th fastest computer in the world and the 10th among educational institutions in the U.S.

4.1. Image denoising. Denoising is an important image processing (IP) step for various image-related applications and often necessary as a pre-processing for other imaging tasks such as segmentation and compression. Thus, image denoising methods have occupied an important position in IP, computer graphics, and their applications. Recently, as the field of IP requires higher levels of reliability and efficiency, various powerful tools of partial differential equations (PDEs) and functional analysis have been successfully applied to image restoration [1, 15, 19, 28, 35, 39, 41, 43, 50] and color processing [8, 20, 29, 31, 44]. In particular, a considerable amount of research has been carried out for the theoretical and computational understanding of the total variation (TV) model [43] and its variants [1, 15, 16, 19, 28, 30, 35, 36, 47].

However, most of those denoising models may lose fine structures of the image due to a certain undesired dissipation. As remedies, the employment of the G-norm [36] and iterative refinement [40] have been studied. But these new methods are either difficult to minimize utilizing the Euler-Lagrange equation approach or have the tendency to keep an observable amount of noise. Recently, in order to overcome the drawbacks, one of the authors suggested the method of nonflat time evolution (MONTE) [18] and the equalized net diffusion (END) approach [17]. The MONTE and END techniques are applicable to various (conventional) denoising models as either a time-stepping procedure or a variant of mathematical modeling.

As another remedy to the undesired dissipation, fourth-order PDE models have emerged [23, 33, 49]. In particular, the Laplacian mean-curvature (LMC) model has been paid a particular attention due to its potential capability to preserve edges of linear curvatures. However, it has been numerically verified [48] that the LMC

model can easily introduce granule-shaped spots to restored images. To overcome the granularity, a hybrid model which combines a TV-based model and the LMC has been proposed [14], briefly described below.

The Laplacian mean-curvature (LMC) model is:

$$\frac{\partial u}{\partial t} + \Delta\kappa(u) = \beta(f-u), \tag{4.1}$$

where $\beta \geq 0$, a constraint coefficient, and $\kappa(u)$ denotes the mean-curvature defined as

$$\kappa(u) = \nabla \cdot \Big(\frac{\nabla u}{|\nabla u|}\Big).$$

In equation (4.1), f is a contaminated image and the solution u represents a denoised image. The LMC model has a major drawback: granularity. The restored image can easily incorporate granule-shaped spots. The LMC model also shows staircasing, a phenomenon that tends to make the restored image locally constant. However, it is relatively easy to cure [29, 35].

As a remedy for the granule-shaped spots introduced by the LMC model, consider the following hybrid model

$$\frac{\partial u}{\partial t} - \sigma\,\widetilde{\kappa}(u) + \Delta\widetilde{\kappa}(u) = \beta(f-u), \tag{4.2}$$

where $\sigma \geq 0$ is a regularization parameter and

$$\widetilde{\kappa}(u) = |\nabla u|\,\kappa(u) = |\nabla u|\,\nabla \cdot \Big(\frac{\nabla u}{|\nabla u|}\Big).$$

Here the gradient magnitude $|\nabla u|$ has been incorporated into $\widetilde{\kappa}(u)$, as a scaling factor, in order to reduce staircasing [35]. The second-order term is introduced for 4.2 to hold a certain degree of maximum principle, with which the model in turn can eliminate the granularity [48].

Equation 4.2 is a *generalized LMC* (GLMC) model with the three model parameters (β, σ and α). The iterations to solve the differential equations involve two extra algorithm parameters: Δt and n. Thus, for a given contaminated image, values have to be selected for these parameters which result in the best restored image. However, when the original uncontaminated image is not known, assessing the quality of the restored image is difficult, if not impossible. In order to gain insight on the influence of these parameters on the quality of the restored image, the model is simulated on known images with synthetically-added Gaussian noise. As a measure of the quality of the restored images, the peak signal-to-noise ratio (PSNR) is adopted. The PSNR is defined as

$$\mathrm{PSNR} \equiv 10\log_{10}\left(\frac{\sum_{ij} 255^2}{\sum_{ij}(g_{ij}-u_{ij})^2}\right)\mathrm{dB},$$

where g denotes the original uncontaminated image and u is the restored image.

Establish uncontaminated image g
Add Gaussian noise to g to produce contaminated image f
Establish parameter counts $\mathrm{N}_\beta, \mathrm{N}_\sigma, \mathrm{N}_\alpha, \mathrm{N}_{\Delta t}, \mathrm{N}_n$
Establish parameter values $\beta[1], \ldots, \beta[\mathrm{N}_\beta]$; $\sigma[1], \ldots, \sigma[\mathrm{N}_\sigma]$;
$\alpha[1], \ldots, \alpha[\mathrm{N}_\alpha]$; $\Delta t[1], \ldots, \Delta t[\mathrm{N}_{\Delta t}]$; $n[1], \ldots, n[\mathrm{N}_n]$
For each combination of $\beta, \sigma, \alpha, \Delta t, n$ values
  Apply denoising procedure on f to produce restored image u
  Calculate PSNR; output $\beta, \sigma, \alpha, \Delta t, n$ and PSNR
End for

FIG. 4.1. *Outline of parametric study for image denoising model*

The parametric study to investigate the influence of the parameters $\beta, \sigma, \alpha, \Delta t$ and n on PSNR is outlined by the the pseudo-code in Figure 4.1. Various plots from the outputs of the study could be produced, including

animations of PSNR as a function of β, σ, α, with either Δt or n fixed and using the other as the variable for the animation.

The number of combinations of parameter values is simply $N_\beta \times N_\sigma \times N_\alpha \times N_{\Delta t} \times N_n$, which could be huge even for small to moderate values of the parameter counts. Fortunately, the denoising procedure can be computed simultaneously for several combinations of the parameters, on a parallel machine. However, the denoising procedure performs nonuniform amounts of computations for each parameter combination; therefore, dynamic load balancing is necessary for efficient utilization of the parallel machine. These characteristics render the parameter study an ideal test application for the load balancing tool described in Section 3. The next two figures summarize the performance of the resulting parallel code for the parameter study with $N_\beta = 9$, $N_\sigma = 9$, $N_\alpha = 9$, $N_{\Delta t} = 9$ and $N_n = 15$, for a total of 98,415 parameter combinations.

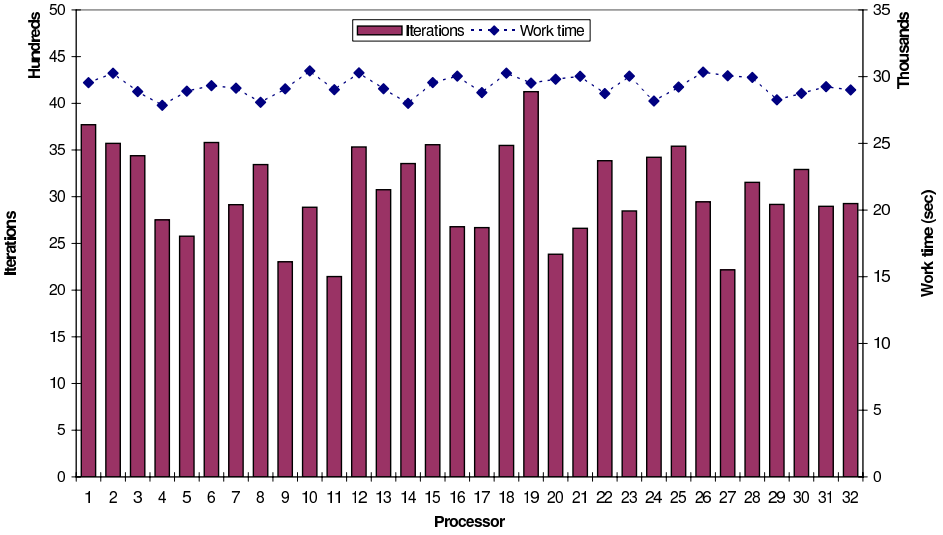

FIG. 4.2. *Distribution of iterations and work times for the parametric study on the image* LenaGray256

Figure 4.2 gives a summary of the performance of the parallel code for the parametric study on the image *LenaGray256*. This study was submitted as a 32-processor job on the EMPIRE cluster; the cluster scheduler assigned homogeneous processors to the job. Since jobs were also executing on the cluster along with the study, the contention for network resources was a source of system-induced load imbalance. However, the major source of load imbalance was the nonuniform amount of computations required by the denoising procedure for different sets of parameter values. The left axis (for the bars) denotes the number of iterations of the loop in Figure 4.1 executed by a processor, while the right axis (for the diamonds) denotes the time in seconds taken by the processor to execute the iterations. The large differences in the number of iterations done by the processors is evidence for application-induced load imbalance. However, the difference between the maximum and minimum work times is only 2581.3 seconds, which is a relatively narrow range. The job time measured by the cluster scheduler was 8.453 hours. An estimate of the sequential cost of the study is 260.4547 hours (∼10.9 days), which is the sum of the work times of the 32 processors. Thus, an estimate of the efficiency is: (estimated sequential cost)/(parallel cost) = (260.4547)/(32×8.453) = 0.963. The high efficiency indicates that the dynamic load balancing tool successfully addressed the load imbalance.

Figure 4.3 gives the summary for the parametric study on the image *BlackCircle*. The cluster scheduler again assigned homogeneous processors to the study, and the job time was 39.546 hours. The differences in iteration counts are significant, indicating the presence of application-induced load imbalance. An estimate of the sequential cost is 1,223.279 hours (∼51 days), which is the sum of the work times of the 32 processors. Thus, an estimate of the efficiency is: (estimated sequential cost)/(parallel cost) = (1223.279)/(32×39.546) = 0.967.

4.2. Vector nonlinear time series. A vector time series is a set of observations of multiple related phenomena across time. The mathematical underpinnings of the statistical analysis of time series incorporate the correlation across time and between series—properties that complicate statistical theory. This is especially true for nonlinear models, where mathematical theory may be extremely difficult, even intractable. Although

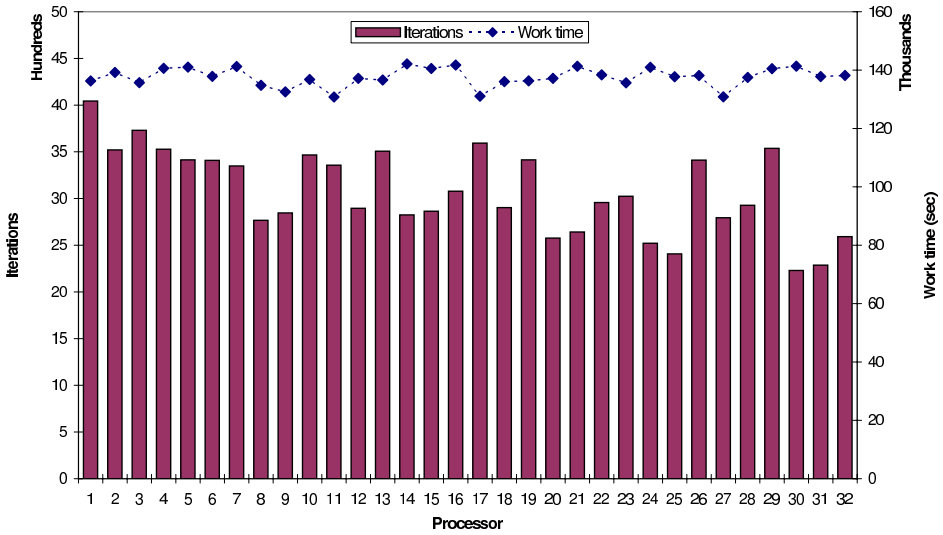

FIG. 4.3. *Distribution of iterations and work times for the parametric study on the image* BlackCircle

complicated in both presentation and theory, vector nonlinear time series are especially useful for describing complicated nonlinear dynamic structures that exists in many time-dependent multivariate series.

Let $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{k,t})'$ denote the vector time series at time $t = 1, 2, \ldots, T$. Then the vector functional coefficient autoregressive model of order p (VFCAR(p)) is defined as

$$\boldsymbol{Y}_t = \boldsymbol{f}^{(0)}(\boldsymbol{Z}_t) + \sum_{j=1}^{p} \boldsymbol{f}^{(j)}(\boldsymbol{Z}_t)\boldsymbol{Y}_{t-j} + \boldsymbol{\varepsilon}_t, \quad t = p+1, \ldots, T, \tag{4.3}$$

where $\boldsymbol{f}^{(j)},\ j = 0, \ldots, p$ are $k \times k$ matrices whose elements are real-valued measurable functions that change as a function of the (possible vector-valued) $\boldsymbol{Z}_t$, and which have continuous second-order derivatives. The error terms $\boldsymbol{\varepsilon}_t$ in (4.3) are such that for each i, the series $\{\varepsilon_{i,t}\}_{t=1}^{T}$ is a white noise sequence, independent of $\{\boldsymbol{Y}_t\}_{t=1}^{T}$. However contemporaneous cross-correlation may exist between $\{\varepsilon_{i,t}\}$ and $\{\varepsilon_{j,t}\},\ i \neq j$. The primary motivation for studying this model is that specific choices for the elements of the $\boldsymbol{f}^{(j)}$ yield parametric models.

The VFCAR(p) model may be considered a hybrid of non-parametric and parametric models since the autoregressive structure is assumed, but there is little or no information about the form of the elements of the $\boldsymbol{f}^{(j)}$. As such, estimation of the parameters of the VFCAR(p) model is done nonparametrically via local regression. Simultaneous estimation of the elements of the $\boldsymbol{f}^{(j)}$ provides improved statistical efficiency when the error terms have positive cross-correlation. In the process of fitting the model (4.3), modified multifold cross-validation is used to determine an optimal bandwidth and value for p by finding the pair of values that minimize the accumulated prediction error. This multistage procedure requires an immense number of arithmetic operations on a univariate series. That number increases exponentially for multivariate series.

The mathematical complexity of the statistical procedures in using the VFCAR model are highly complicated, necessitating the use of simulation to study their properties. Consequently, Monte Carlo simulation is often relied upon to give direction, to interpret, and to explain complex analytical results. In general, it can be said that as the number of Monte Carlo replications increases, the result of the simulation approaches the "truth". The more complex the structure, the larger the number of replications needs to be.

The examination of statistical properties of methods related to the VFCAR models via simulations on a single processor would require execution times of a few weeks or even months. Statisticians have been known to base empirical results on a relatively small number of simulation replications, sacrificing precision, accuracy, and possibly compromising the reliability of results in the interest of time. As a result, techniques which may require thousands of replications for accuracy and reliability often have at most, a few hundred. Fortunately, the replications are amenable for computation in parallel. Thus, parallel processing technology can be exploited to enable the extensive simulation of a variety of models within reasonable running time limits.

```
Input model specifications; no_reps, nh, no_bs_reps
...
DO rep_no=1,no_reps  ! replications
  ...
  DO i1=1,nh+1       ! bandwidth
    ...
  END DO
  ...
  DO i2=1,no_bs_reps ! bootstrap test
    ...
  END DO
  ...
END DO
...
```

FIG. 4.4. *Outline of VFCAR simulation*

Figure 4.4 gives a high-level outline of the simulation procedure to investigate the statistical properties of estimators and hypothesis tests related to the VFCAR model for multivariate nonlinear time series. Results of computational experiments in which the simulation procedure is treated as a 1D-loop (i. e., only the replication loop is parallelized) are reported in [3, 2]. Estimated efficiencies of up to 97% were achieved on 64 processors.

As a further demonstration of the capability of the dynamic load balancing tool, the simulation was treated as a 2D-loop. Therefore, a parallelization strategy similar to Figure 3.4 was followed. The 2D-loop version of the application, with `no_reps`=10000, `nh`=50, and `no_bs_reps`=500, was submitted as a 64-processor job to the EMPIRE cluster. The adaptive factoring technique [4, 6] for loop scheduling was specified. Owing to the small amount of concurrency in the inner (bandwidth and bootstrap test) loops, `maxSize` was set to 4. In the sample run summarized by Figure 4.5, the cluster scheduler assigned to the job 2, 10, 10 and 42 processors from racks 4, 5, 8 and 11, respectively. Rack 11 contains 1.266GHz processors, while the other racks contain 1.0GHz processors. Eighteen (18) `iComm`s were formed: `iComm`s 2–6 and 12–15, each with 4×1.266GHz processors; `iComm` 1 with 3×1.266GHz processors; `iComm` 16 with 2×1.266GHz processors; `iComm`s 7–10, each with 4×1.0GHz processors; `iComm`s 11, 17, and 18, each with 2×1.0GHz processors; and the remaining processor as the `coordinator`.

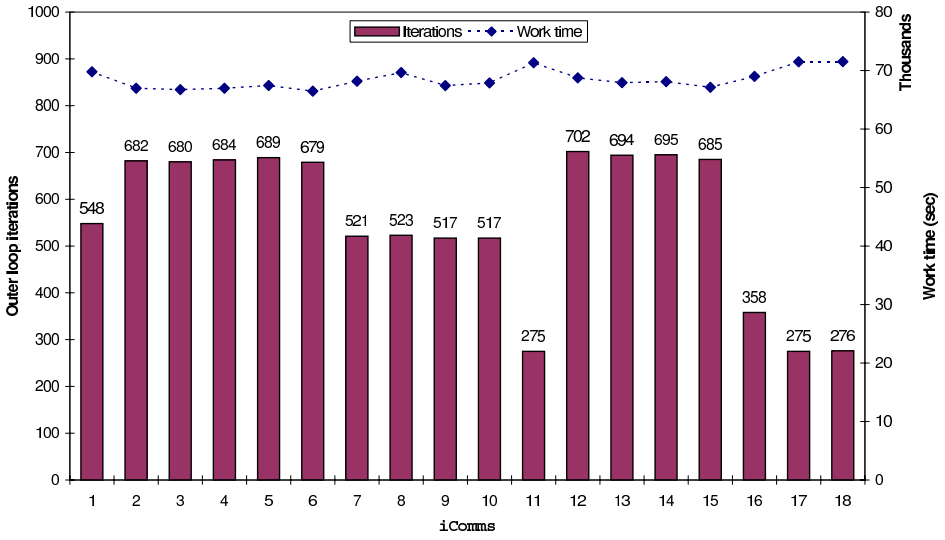

FIG. 4.5. *Distribution of outer loop iterations and work times for VFCAR simulation*

Figure 4.5 illustrates the number of replications (outer loop iterates) executed by each `iComm` and their corresponding work times. The heterogeneity of the `iComm`s is reflected in unequal numbers of replications

executed. But the variation in the work times is small: the average of the absolute deviation (AAD) of the work times is only 2% of the mean work time, computed as follows: if x_i is the work time of the ith `iComm` and $\bar{x}$ is the mean work time, then AAD=$\sum_{i=1}^{18} |x_i - \bar{x}|/(18 * \bar{x})$. The parallel job time measured by the cluster scheduler is 19.89 hours. An estimate of the sequential cost is 1193.06 hours (∼49.71 days), obtained as the sum of the costs from each `iComm`. Thus, an estimate of the efficiency is: (estimated sequential cost)/(parallel cost) = (1193.06)/(64*19.89) = 0.937.

Note that the estimated efficiency of the 2D-loop version (93.7%) of the simulation is slightly less than that of the 1D-loop version (97%). This is to be expected since the overhead of the 2D-loop version is higher. In any case, the results are indicative of the effectiveness of the two-level dynamic load balancing strategy implemented by the tool.

5. Summary and Future Work. An interesting load balancing problem arises when running a scientific application containing CPU-intensive one-dimensional or two-dimensional parallel loops on a general-purpose cluster: the loop iterates may have nonuniform execution times, and the processors allocated to the application may be heterogeneous and may reside on different cluster racks. These sources of load imbalance may not be known or predictable before the application starts.

This paper describes a dynamic loop scheduling tool to address the above problem. The tool is especially designed for computational investigators who have little familiarity with developing message-passing programs. The tool easily integrates into an existing sequential application, after minor code modifications, to produce a message-passing version. For a 2D-loop, the tool follows a two-level strategy, where at one level, chunks of iterates of the outer loop are executed concurrently, and simultaneously at another level, chunks of iterates of the inner loop are also executed concurrently.

The tool has been integrated into nontrivial sequential applications. The applications mentioned in this paper achieved estimated efficiencies in excess of 90% on a general-purpose heterogeneous cluster. The tool is being integrated into other scientific applications; discoveries made through these applications will be reported elsewhere. The use of the tool for dynamic scheduling of 3D-loops (or even higher) is being investigated. Intuitively, a 3D-loop can be implemented using the same two-level strategy, where a 1D-loop is executed at one level, while the 2D-loop is executed at the other level. Results of these investigations will be reported in the future.

Acknowledgments. We thank Hyeona Lim, Neil Williams and Seongjai Kim for their collaboration on the simulation of a hybrid model for image denoising. We also thank Jane Harvill and John Lestrade for their collaboration on the analysis of gamma-ray burst datasets using vector functional coefficient autoregressive time series models.

REFERENCES

[1] L. Alvarez, P. Lions, and M. Morel, *Image selective smoothing and edge detection by nonlinear diffusion. II*, SIAM J. Numer. Anal., 29 (1992), pp. 845–866.

[2] I. Banicescu, R. L. Cariño, J. L. Harvill, and J. P. Lestrade, *Computational challenges in vector functional coefficient autoregressive models*, in Lecture Notes in Computer Science 3514, Computational Science—ICCS 2005, V. S. et al., ed., Springer-Verlag, 2005, pp. 237–244.

[3] ———, *Simulation of vector nonlinear time series on clusters.*, in Proceedings of the 19th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2005, p. CDROM.

[4] I. Banicescu and Z. Liu, *Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes*, in Proceedings of the High Performance Computing Symposium (HPC) 2000, 2000, pp. 122–129.

[5] I. Banicescu and V. Velusamy, *Performance of scheduling scientific applications with adaptive weighted factoring*, in Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium—10th Heterogenous Computing Workshop (IPDPS-HCW 2001) CDROM, IEEE Computer Society, Apr. 2001.

[6] ———, *Load balancing highly irregular computations with the adaptive factoring*, in Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium—11th Heterogeneous Computing Workshop (IPDPS-HCW 2002) CDROM, IEEE Computer Society, 2002.

[7] I. Banicescu, V. Velusamy, and J. Devaprasad, *On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring*, Cluster Computing: The Journal of Networks, Software Tools and Applications, 6 (2003), pp. 215–226.

[8] P. Blomgren and T. Chan, *Color TV: Total variation methods for restoration of vector valued images*, IEEE Trans. Image Process., 7 (1998), pp. 304–309.

[9] O. A. R. Board, *OpenMP specifications*, 2003. `http://www.openmp.org/specs`

[10] J. M. Bull, *Feedback guided dynamic loop scheduling: Algorithms and experiments*, in Lecture Notes in Computer Science

1740, Proceedings of the 4th International Euro-Par Conference (EuroPar'98), D. Pritchard and J. Reeve, eds., Springer-Verlag, 1998, pp. 377–382.

[11] R. L. Cariño and I. Banicescu, *Dynamic scheduling parallel loops with variable iterate execution times*, in Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium - 3rd Workshop on Parallel and Distributed Scientific and Engineering Computing With Applications (IPDPS-PDSECA 2002) CDROM, IEEE Computer Society, 2002.

[12] ———, *Load balancing parallel loops on message-passing systems*, in Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), S. Akl and T. Gonzales, eds., ACTA Press, 2002, pp. 362–367.

[13] ———, *A load balancing tool for distributed parallel loops*, Cluster Computing, 8 (2005), pp. 313–321.

[14] R. L. Cariño, I. Banicescu, H. Lim, N. Williams, and S. Kim, *Simulation of a hybrid model for image denoising*, in Proceedings of the 20th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2006, p. (to appear).

[15] F. Catte, P. Lions, M. Morel, and T. Coll, *Image selective smoothing and edge detection by nonlinear diffusion.*, SIAM J. Numer. Anal., 29 (1992), pp. 182–193.

[16] Y. Cha and S. Kim, *Edge-forming methods for color image zooming.* (accepted to IEEE Trans. Image Process.).

[17] ———, *Equalized net diffusion (END) for the preservation of fine structures in PDE-based image restoration.* (submitted to IEEE Trans. Image Process.).

[18] ———, *MONTE: The method of nonflat time evolution in PDE-based image restoration.* (submitted to IEEE Trans. Image Process.).

[19] T. Chan, S. Osher, and J. Shen, *The digital TV filter and nonlinear denoising*, Technical Report #99-34, Department of Mathematics, University of California, Los Angeles, CA 90095-1555, October 1999.

[20] T. Chan and J. Shen, *Variational restoration of non-flat image features: Models and algorithms*, SIAM J. Appl. Math., 61 (2000), pp. 1338–1361.

[21] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan, *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*, Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377 `http://www.cs.sandia.gov/Zoltan/ug_html/ug.html`

[22] M. Forum, *The Message Passing Interface Standard*, 1995. `http://www-unix.mcs.anl.gov/mpi`

[23] J. Greer and A. Bertozzi, *Traveling wave solutions of fourth order pdes for image processing*, SIAM J. Numer. Anal., 36 (2004), pp. 38–68.

[24] D. Hancock, R. Ford, T. Freeman, and J. Bull, *An investigation of feedback guided dynamic scheduling of nested loops*, in Proceedings of the 2000 International Conference on Parallel Processing Workshops (ICPPW'00), IEEE Computer Society, 2000, pp. 315–321.

[25] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, *Load-sharing in heterogeneous systems via weighted factoring*, in Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures (SPAA 1996), 1996, pp. 318–328.

[26] S. F. Hummel, E. Schonberg, and L. E. Flynn, *Factoring: A method for scheduling parallel loops*, Communications of the ACM, 35 (1992), pp. 90–101.

[27] G. Karypis and V. Kumar, *ParMETIS: Parallel graph partitioning and sparse matrix ordering.* `http://www-users.cs.umn.edu/~karypis/metis/parmetis/index.html` 2003.

[28] S. Kim, *Edge-preserving noise removal: Motion by mean curvature.* (submitted to SIAM J. Sci. Comput.).

[29] ———, *PDE-based image restoration: A hybrid model and color image denoising.* (accepted to IEEE Trans. Image Process.).

[30] S. Kim and H. Lim, *A non-convex diffusion model for simultaneous image denoising and edge enhancement.* (submitted to Electron. J. Differ. Equ.).

[31] R. Kimmel and N. Sochen, *Orientation diffusion or how to comb a porcupine?*, Special issue on PDEs in Image Processing, Computer Vision, and Computer Graphics, J. Visual Comm. Image Representation, 13 (2002), pp. 238–248.

[32] C. Kruskal and A. Weiss, *Allocating independent subtasks on parallel processors*, IEEE Transactions on Software Engineering, SE-11 (1985), pp. 1001–1016.

[33] M. Lysaker, A. Lundervold, and X. Tai, *Noise removal using fourth-order partial differential equations with applications to medical magnetic resonance images in space and time*, IEEE Trans. Image Process., 12 (2003), pp. 1579–1590.

[34] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale, *DRAMA: A library for parallel dynamic load balancing of finite element applications*, in Euro-Par '99: Proceedings of the 5th International Euro-Par Conference on Parallel Processing, London, UK, 1999, Springer-Verlag, pp. 313–316.

[35] A. Marquina and S. Osher, *Explicit algorithms for a new time dependent model based on level set motion for nonlinear deblurring and noise removal*, SIAM J. Sci. Comput., 22 (2000), pp. 387–405.

[36] Y. Meyer, *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations*, vol. 22 of University Lecture Series, American Mathematical Society, Providence, Rhode Island, 2001.

[37] S. Microsystems, *Forte C 6 /Sun Workshop 6 Compilers C User's Guide*, 2003. `http://docs.sun.com/source/806-3567/parallel.html`

[38] ———, *Fortran User's Guide*, 2003. `http://docs.sun.com/source/806-3591/E_directives.html`

[39] M. Nitzberg and T. Shiota, *Nonlinear image filtering with edge and corner enhancement*, IEEE Trans. on Pattern Anal. Mach. Intell., 14 (1992), pp. 826–833.

[40] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, *Using geometry and iterated refinement for inverse problems (1): Total variation based image restoration*, CAM Report #04-13, Department of Mathematics, UCLA, LA, CA 90095, 2004.

[41] P. Perona and J. Malik, *Scale-space and edge detection using anisotropic diffusion*, IEEE Trans. on Pattern Anal. Mach. Intell., 12 (1990), pp. 629–639.

[42] C. Polychronopoulos and D. Kuck, *Guided self-scheduling: A practical scheduling scheme for parallel supercomputers*, IEEE Transactions on Computers, C-36 (1987), pp. 1425–1439.

[43] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D, 60 (1992), pp. 259–268.
[44] N. Sochen, R. Kimmel, and R. Malladi, *A general framework for low level vision*, IEEE Trans. Image Process., 7 (1998), pp. 310–318.
[45] T. Tabirca, L. Freeman, S. Tabirca, and L. T. Yang, *Feedback guided dynamic loop scheduling; a theoretical approach*, in Proceedings of the 2001 International Conference on Parallel Processing Workshops (ICPPW'01), IEEE Computer Society, 2001, pp. 115–121.
[46] T. H. Tzen and L. M. Ni, *Trapezoid self-scheduling: A practical scheduling scheme for parallel computers*, IEEE Transactions on Parallel Distributed Systems, 4 (1993), pp. 87–98.
[47] L. Vese and S. Osher, *Numerical methods for p-harmonic flows and applications to image processing*, Technical Report #01-22, Department of Mathematics, University of California, Los Angeles, CA 90095, USA, August 2001. (To appear in SIAM Numer. Anal.).
[48] N. Williams, H. Lim, and S. Kim, *Numerical schemes for the Laplacian mean-curvature flow and their applications to image denoising*. (In preparation).
[49] Y. You and M. Kaveh, *Fourth-order partial differential equations for noise removal*, IEEE Trans. Image Process., 9 (2000), pp. 1723–1730.
[50] Y. You, W. Xu, A. Tannenbaum, and M. Kaveh, *Behavioral analysis of anisotropic diffusion in image processing*, IEEE Trans. Image Process., 5 (1996), pp. 1539–1553.

Edited by: Dana Petcu
Received: May 29, 2007
Accepted: June 18, 2007