Observations on Data Distribution and Scalability of Parallel and Distributed Image Processing Applications

Roman Pfarrhofer and Andreas Uhl
uhl@cosy.sbg.ac.at

Outline

- Introduction
- MATLAB Custer Computing: MDICE
- Distributed Search in Image Databases
  - Template Matching
  - Experimental Settings
  - Experimental Results
- Conclusions

Introduction

- Creation, processing, and management of visual data require an enormous computational effort, often too high for single processor architectures.
- Inherent data parallelism in visual data makes image and video processing natural application areas for parallel computing.
- "Workshop on Parallel and Distributed Image Processing, Video Processing, and Multimedia (PDIVM)" @ IPDPS
- Software based approaches are becoming more popular in this area because of
  - performance increase of general-purpose processors (media processors)
  - rapid evolution of multimedia techniques which has dramatically shortened the time available to come up with a new hardware design for each improved standard or technique.

Cluster Computing and MATLAB

- Emerge of cluster computing → a cheap and highly available and yet powerful architecture for image processing applications

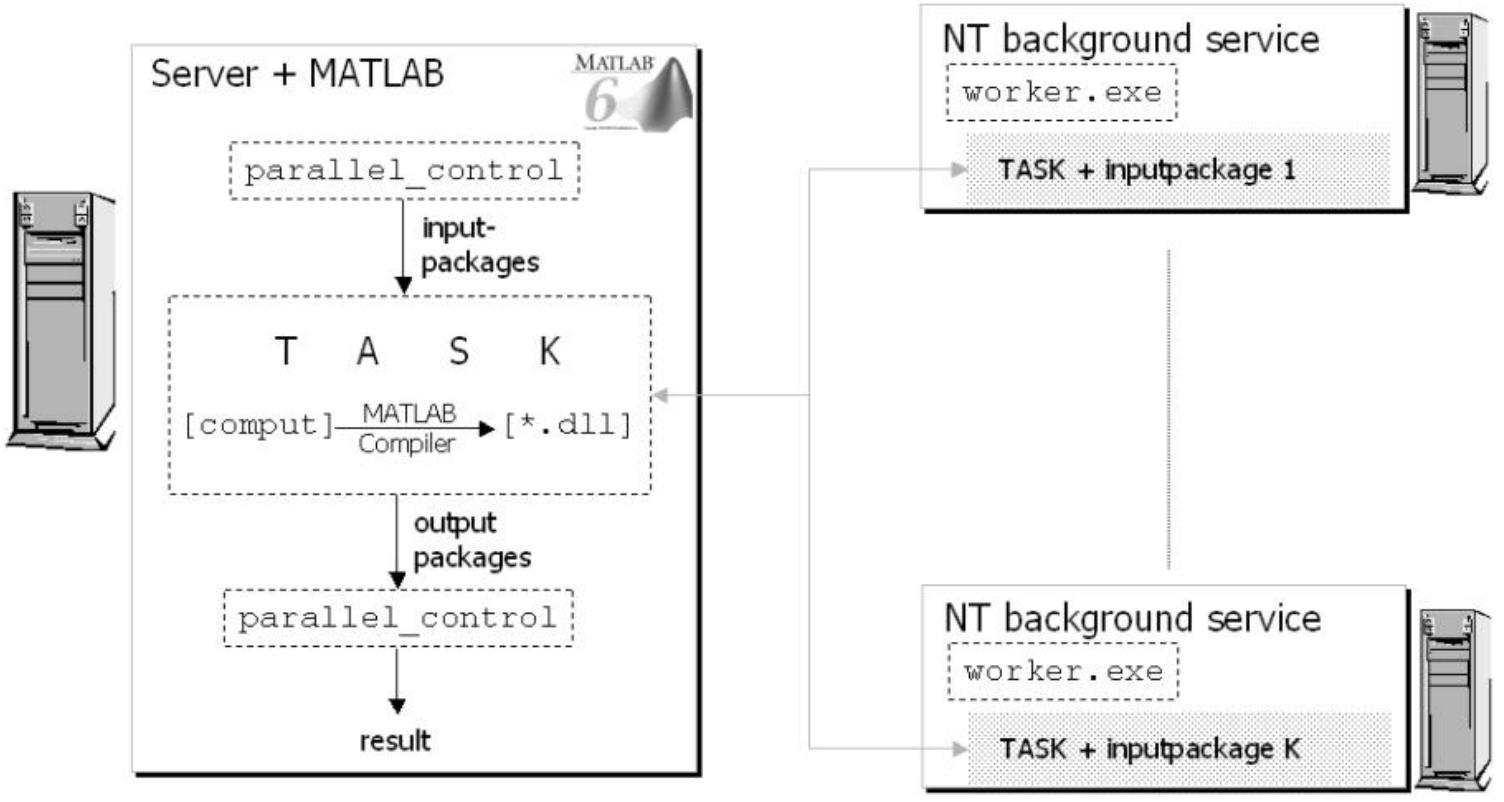
- MATLAB provides users with easy access to an extensive library of high quality numerical routines which can be used in a dynamical way and may be easily extended and integrated into existing applications.

- MATLAB in HPC
  - Developing a high performance interpreter (MPI/PVM based communication routines) or coarse grained parallelization by splitting up work among multiple MATLAB sessions.
  - Calling high performance numerical libraries (e.g. SCALAPACK)
  - Compiling MATLAB to another language (e.g. C, HPF)

MDICE

- MDICE: **M**ATLAB-based **DI**stributed **C**omputing **E**nvironment
- Goal: in contrast to previous approaches the goal is to get along with a single MATLAB client (instead of a MATLAB client at each participating compute node)
- The main idea was to change the client program in a way that it can be compiled to a standalone application.
- The compiled client program library and the datasets for the job are sent to the compute node, where a background client service is running with low priority.
- For this reason the involved client machines may be used as desktop machines by other users during the computation.

Client-Server Concept of MDICE



Template Matching

- An image database is searched for a specific template.
- A varying amount of noise is added the template and rotated versions are created (partially at the server).
- The templates are duplicated on each client.
- At the client, for all templates in the search area a normalized cross-correlation is computed and the highest value is recorded.

Data Distribution

- The image database is partitioned into a user-selected number of jobs and the jobs (i.e. images or image tiles) are distributed to be processed independently on the clients.
- If the number of jobs exceeds the number of images, these need to be tiled accordingly.
- To guarantee correct results the images need to be distributed in a redundant way where the size of the overlap has to be the width of the template-1 in each dimension.
- Alternative: exchange the required border data among the clients $\longrightarrow$ high communication cost, especially on a cluster

Experimental Settings

- Computational task is split into a certain number of equally sized jobs N to be distributed by the server among the M clients in a dynamic fashion ("asynchronous single task pool method")

- Server machine (1.99 GHz Intel Pentium 4, 504 MB RAM) and client machines (996 MHz Intel Pentium 3, 128 MB RAM), both types under Windows XP Prof.

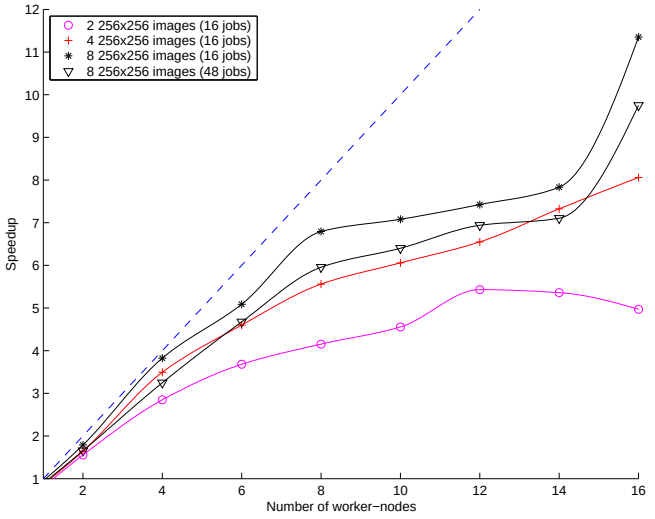
- 100 MBit/s Ethernet network.

- Sequential referece times have been achieved on a 996 MHz client machine with a compiled (not interpreted) version of the application to allow a fair comparison

- We use MATLAB 6.5.0 with the MATLAB compiler 3.0 and the LCC C compiler 2.4.

Experimental Results I: Speedup with Varying Problem Size

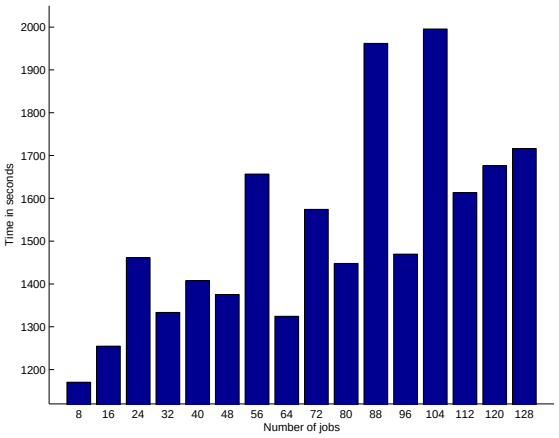
- We notice plateaus resulting from load distribution problems (e.g., 16 jobs may not be efficiently distributed among 14 clients)
- Lower speedup and less pronounced plateaus are exhibited in case of smaller problem size (initial communication phase dominates other problems).



- The reason for the increased speedup for more data is the decreasing amount of overlapping data required for redundant tiling (e.g. 2 images are tiled into 8 tiles each whereas 8 images are cut into halves only) $\longrightarrow$ smaller amount of overlapping data reduces computation effort. The reason is NOT improved computation/communication ratio !
- 48 jobs instead of 16 jobs allow better balanced load BUT a higher amount of overlapping data and the associated computations $\longrightarrow$ performance is lower.

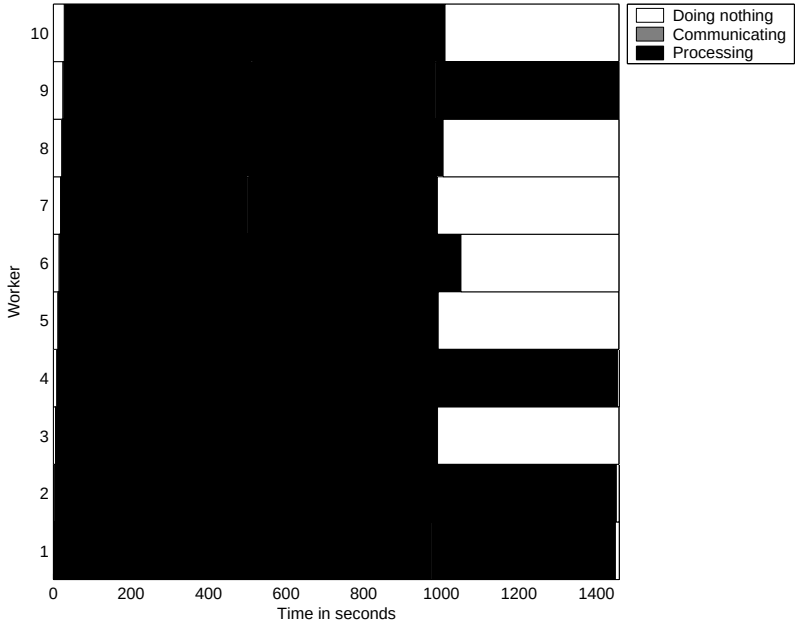
Experimental Results II: Time Demand with Varying Number of Jobs (10 clients, 8 512^2 Images)

- No improved performance for a larger number of jobs due to improved load balancing (more jobs cause a large amount of overlapping data in the tiling $\rightarrow$ higher amount of overall computations).
- The number of jobs devided by the number of images in the database gives the number of tiles required per image. If the resulting number is a prime, the image is tiled along one dimension only which results in a higher amount of overlapping data as in case of tiling along both dimensions.

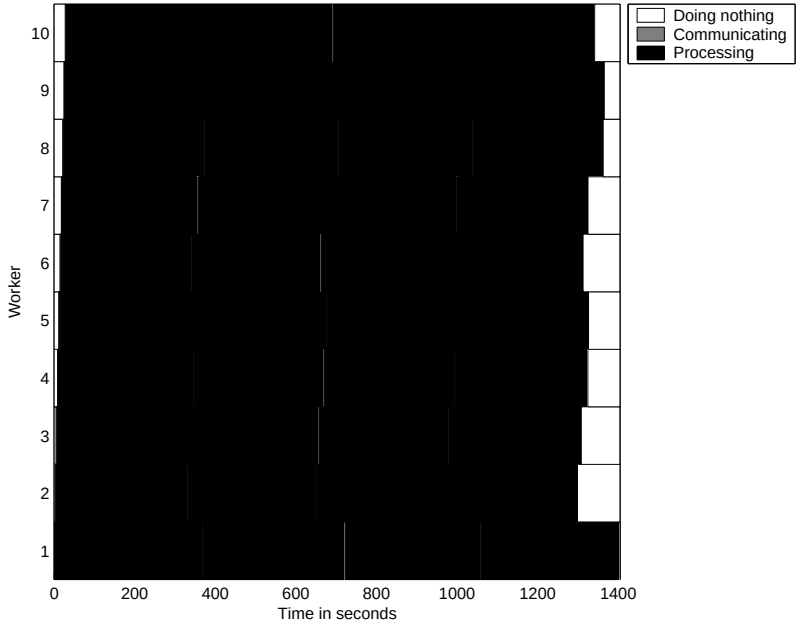


- Poor performance is especially exhibited for 24, 40, 56, 88, and 104 jobs (which corresponds to 3, 5, 7, 11, and 13 tiles per image, respectively).
- Using 24 jobs as compared to 40 jobs should be faster due to the lower amount of overlapping data. The 40 job setting is faster (perfect load distribution compensates higher amount of computation and communication)

Experimental Results III: Execution Visualization



(a) 24 jobs on 10 clients



(b) 40 jobs on 10 clients

Conclusions

1. Increasing the number of jobs (i.e. image tiles to be processed) does not only increase the communication amount but also the computational effort is increased since the amount of overlapping data grows. This has two major implications:
   (a) Increasing the number of jobs to achieve better balanced load might be contraproductive and result in worse performance.
   (b) Increasing the number of processing elements when keeping the problem size fixed does not lead to better performance at some point (poor scalability).

2. The question how the image data is partitioned is crucial with respect to performance in case the application involves neighbourhood operations. Simple partitioning along one dimension only (row partitioning) is shown to perform significantly worse as compared to tiling both image dimensions.