Performance Optimisation of a Lossless Compression Algorithm using the PACE Toolkit

J.D. Turner, R. Lopez-Hernandez, D.J. Kerbyson, G.R. Nudd
High Performance Systems Group, University of Warwick, Coventry, UK
{jdt,roberto,djke,grn}@dcs.warwick.ac.uk

Abstract

With the large increase in interest and research in distributed systems, the need for performance prediction and modelling of such systems has become important to decrease the system's complexity. One such prediction technique is PACE, a series of performance modelling tools developed at Warwick, which allows a user to create very accurate performance models of sequential and distributed systems. Such a model can be used in an application steering method, where a performance model is added to the overall system such that the application can be optimised to match a particular constraint. To describe this method, a lossless compression technique, also developed at Warwick, is optimised to fit a distributed system where a time constraint is necessary. It is shown that the compression ratio gained by the compression technique grows, as expected, while the time constraint given to the application becomes more relaxed.

1 Introduction

With the major advances in high performance systems and networking technologies, the emphasis in high performance has changed from large multi-processor arrays to distributed systems, where individual computer systems communicate over a powerful network. These distributed systems, an example of which is the Information Power GRIDS [3], allow a particular application to be executed on theoretically an infinite number of systems with different hardware and software components. The complexity involved in such a distributed system raises dramatically however, when more and more applications are executed on such a system.

Within the GRIDS system, this complexity is reduced with the use of performance information, which allows application execution to be tuned to gain the best performance for the distributed system available. Such performance measurement tool-sets have been developed, which aim to aid this performance prediction, such as Falcon [6], Paradyn [7], and Pablo [2]. However, these tools are mainly focused on the analysis of the performance after the execution has taken place, to allow performance-hindering elements, such as bottlenecks that may be inherent within the application, to be discovered.

The performance modelling toolset being developed at Warwick focuses more on the performance prediction aspects of the system. PACE (Performance Analysis and Characterisation Environment) [8, 9] allows a performance model of the system to be designed, such that accurate measurements of the application's execution behaviour can be determined prior to the actual execution itself. These predictions can then be used to optimise the execution across the distributed system.

An example of such an application that could be optimised for such as system is the Spatial and Motion Compression (SMC) algorithm, a lossless technique, which has also been developed at Warwick. Although lossly techniques are frequently used in applications such as DVD, where complete reconstruction is not necessary, there are an increasing number of applications where total reconstruction, and high compression ratios, are necessary. Examples of such are in professional video postproduction, where lossy techniques would introduce errors within the video stream that would be visible to the trained eye, even at the cost of a low compression ratio. Another example is within the medical imagery sector, where it is necessary to compress thousands of images but imperative to keep the high quality of the image.

The method of optimisation chosen for this compression technique is to constrain the execution to a particular time constraint, such that the best compression ratio can be achieved for a given execution 'slot' within the distributed system. This optimisation was implemented using an Application Steering method [1], which allows the PACE performance model to be included within the application implementation. The performance model then specifies the system parameters to the application such that any constraint is met.

In this paper, a detailed description of the performance modelling techniques available within the PACE toolkit, and the SMC lossless compression technique are found within Sections 2 and 3 respectively. Section 4 covers the application steering methods used, with emphasis placed on the implementation of the performance model, with Section 5 showing the results achieved.

2 Performance Modelling using PACE

The PACE performance modelling toolkit devised at the University of Warwick allows accurate performance predictions on sequential, parallel, and distributed systems executed on a vast array of hardware architectures. PACE was designed such that a performance model of a system can be implemented, and used to gain accurate predictions and results of such a system, without requiring the person implementing the model to have a detailed knowledge of the theory behind performance modelling. Once the model has been implemented, then a vast array of accurate predictions can be found, but more importantly, predictions on various sequential and distributed hardware systems.

The PACE toolset is based on a layered, characterisation framework, which separates the software and hardware elements of the system with parallel templates, as shown in Figure 1. Each layer has one or more associative, modular objects that describe the individual sections of both the software and hardware components. The modularity of the design of these objects is such that they can be used for many different application predictions. For example, the same hardware object can be used with any model which is executed on that system. Once these objects have been defined, they are linked together using the CHIP^{3}S compiler to create the performance model. PACE includes a variety of tools which allows the user to find the required information from this model within the run-time environment. The use of the layered framework, the various object definitions, and the various results that can be predicted from the model are described within this section.

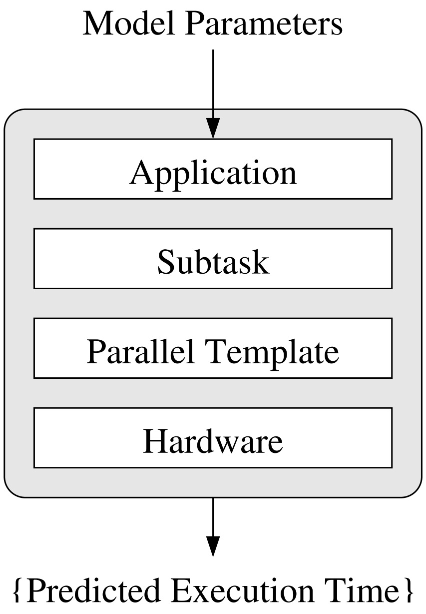

Figure 1. The PACE framework

2.1 The Characterisation Framework and Object Definition

To accurately predict the performance of both distributed and sequential systems, a three-layer approach is used, with a layer for the software, parrallelisation, and hardware parts of the system. The purpose of such layers are described below.

2.1.1 Application Layer

The application layer is the interface between the user and the rest of the model. It is here where the global parameters to the system are defined, and where the individual subtasks are predicited at the required times. The global parameters are also altered as need be, and passed to the correct subtasks such that the prediction of each individual subtask is as accurate as possible.

Each performance model has only one application object.

2.1.2 Subtask Layer

The subtask layer is where all the main sequential parts of the system, which may be parallelised, are defined. The subtask objects which reside within this layer each represent one of these sequential parts, and describe the sequential execution dependent upon the parameters passed to the subtask from the application object. There are likely to be many subtask objects within the performance model.

Each subtask object is made up of two main sections. The first is where all the parameters for the whole subtask are defined, which normally include parameters passed down from the application object. These parameters affect the execution time of the prediction code; which is the second main section of the subtask. Every function associated with the sequential execution of the system being modelled is converted into a ‘cflow’ statement. Each ‘cflow’ statement represents the flow of instructions executed on a certain hardware platform, and reads from the hardware layer the correct information within the hardware object to predict the time that each of these instructions will take to execute. ‘cflow’ statements may call other ‘cflow’ statements within the same subtask; as functions may call other functions within a C program.

2.1.3 Parallel Template Layer

The parallel template layer describes the parallel characteristics of subtasks where the communication and computation interactions between processors is concerned. Each template reads the required information from the hardware objects to predict the parallel and communication time across the processors. There may be many parallel template objects within the model, with each one describing the parallel patterns concerned with a specific subtask. A specific parallel template for sequential execution is also available.

Each parallel template object is made up of individual steps, which either corresponds to an access to one or more of the available hardware resources, whether it be computation on a CPU or communication across the network. This execution is defined using a 'step' command, which defines the hardware resource that will be used and any necessary parameters. By evaluating each step, the PACE run-time system calls the appropriate hardware model and returns the execution time for each step. Thus an accurate prediction for the parallel execution of the model is found.

2.1.4 Hardware Layer

The hardware layer contains the hardware objects which describe the computational and network communication parameters associated with each hardware platform. The number of hardware objects is dependant on the different hardware platforms that the system is executed upon.

Each hardware object contains the time taken for every possible sequential execution, every possible parallel communication, and the possible results of the cache, for that particular hardware device. Each of these parameters are defined within the object, and when necessary, called upon by either the appropriate subtask object (for sequential execution), or the appropriate parallel template object (for parallel communication).

An accurate hardware object is created automatically with a PACE benchmarking tool. Depending on the hardware system, this tool times the execution time for a large number of certain micro-operations; the average of which is stored within the hardware object. The benchmarking tool outputs a standard deviation of the execution times found, which gives the user a measure of the accuracy of the measurements, and the choice to re-create the object.

2.2 The PACE Run-time Environment

Once a model containing a number of objects has been designed and implemented, it is necessary to compile and link them together to obtain the performance model executable. This model can then be used for a number of different modelling techniques to find accurate performance information of the system.

Each object is compiled using the CHIP3S compiler 'chip3s', whose outputs are linked together using the linker 'chip3sld'. An example of which may look like this:

```
bash% chip3s -o app.o app.la
bash% chip3s -o sub.o sub.la
bash% chip3s -o par.o par.la
bash% chip3sld -o model app.o sub.o par.o
```

which would create an executable performance model called 'model'. Simply executing this model would then output the performance prediction of the time taken to execute the application for the default set of input parameters defined within the application object. These input parameters can naturally be changed to the user's discretion. For example

```
bash% ./model Nframe=5
```

will output the execution time if the parameter 'Nframe' was set to five instead of the model's default value.

More detailed information can be obtained from the model with various options which are inherent in the compiled model. Of the more important options are the '-debug' option, which outputs a detailed prediction of all communication and execution times that the model produces. This option is very powerful, and allows the user to really focus on the required area.

3 SMC Compression

SMC (Spatial and Motion Compression) is a lossless compression technique developed at the University of Warwick, which can be applied to the compression of computer generated animation sequences. SMC combines several lossless compression techniques. It exploits the spatial and temporal redundancies found in computer animations. Consecutive animation frames are very similar. Most of the objects in one frame appear again in the next frame with slightly different positions and orientations. This kind of similarities or redundancies can be exploited using a temporal compression technique. However, there might be new information that cannot be inferred from previous frames. For example, an object that was occluded by other object that is moving, new information introduced on one side of the image when the camera is panning, new image details appearing when the camera is zooming in, a face that was occluded on a rotating object, etc. This information can sometimes be better encoded with a spatial compression technique, using neighbouring information in the same frame. In particular, a spatial compression technique exploits the redundancies found in areas filled with the same colour or areas with smooth changes of colour.

In this section all the components of the SMC lossless compression technique for computer animation sequences will be detailed. SMC stands for Spatial and Motion Compression - it combines spatial compression techniques with temporal compression techniques in order to obtain good compression results. SMC is an asymmetric lossless compression technique. This is, the compression can be, depending on the compression parameters, slightly faster or much slower than the decompression. It has been found that good compression results can be obtained with fast options. However, slow compression options are available to provide the option if higher compression ratios are required.

SMC uses only input pixel information RGB or RGBA (Red, Green, Blue and Alpha channels) from each frame in a sequence, thus not relying on the information contained within the animation scripts [12]. It has been implemented as a software library and can be easily incorporated into other software packages. Compression ratios of 4:1 and higher can be obtained using this technique.

3.1 Overview of SMC Lossless Compression

A schematic view of the SMC compression technique is shown in Figure 2. The main part of this compression technique consists of a loop in which the spatial and the temporal compression techniques are applied. This loop is performed for the number of reference frames specified, from 0 (no reference frames) up to n previous frames. The main processing stages contained within SMC are as follows:

1. Initially the current frame (frame i) is coded using a spatial compression operation (resulting in a prediction image labelled S in Figure 2.).

2. For each reference frame (frames i-1, i-2, i-3, ...), block movement is calculated resulting in a three-element motion vector for each block - a position movement vector (x, y) and also a z component indicating the reference frame in which the matching block can be found.

3. For each block in the image, the best matched block is selected - either from the spatial prediction or from one of the motion calculated frames.

4. A residual is calculated by the difference between the frame constructed (with spatial and motion blocks) and the original frame. This residual combined with the motion vectors is all the information necessary to allow lossless recontruction later.

5. Redundant motion vectors are then removed. These occur since the blocks encoded using the spatial compression technique do not have any motion information.

6. The alpha channel and the vector component Z are encoded using a block area encoding technique.

7. Finally the residual and the vector information are further encoded using an entropy coder. There are two options available in this case. The fastest one is a rice entropy encoder. The second option is a rice entropy encoder followed by an arithmetic encoder.

Each of these stages contained in SMC are explained in more detail below.

3.2 Spatial Encoding

The spatial compression technique incorporated in SMC is the spatial predictor used in LOCO [10]. CALIC [11] is the best spatial coder available today. However, LOCO uses less prediction rules and therefore is faster. These two techniques can detect object edges in the images. Consequently, they usually outperform DPCM, which is the current lossless mode of the JPEG standard.

In a spatial prediction technique, the encoder and decoder perform the same prediction with information previously sent or received (respectively). The predicted value is then corrected by adding a residual. The residual is the difference between the original frame and the predicted frame. The residual image is the only information required for the reconstruction of the original image by the decoder. Therefore, only the residual needs to be encoded.

In SMC, the spatial compression technique is applied before applying any other compression operation. Hence, the LOCO predictor is applied to the current animation frame and a prediction image is generated. The prediction image consists of the predicted pixels produced by LOCO. This is done on an individual pixel by pixel basis. A residual image can be obtained from the difference between the original frame and the prediction image. However, the temporal compression technique must be applied first before calculating the final residual.

The spatial compression technique is given preference over the temporal compression and is applied first, as it does not require any further information such as motion vectors apart from the residual image.

3.3 Motion Estimation

A temporal compression technique is applied to the animation frames, after the spatial compression step. In SMC,

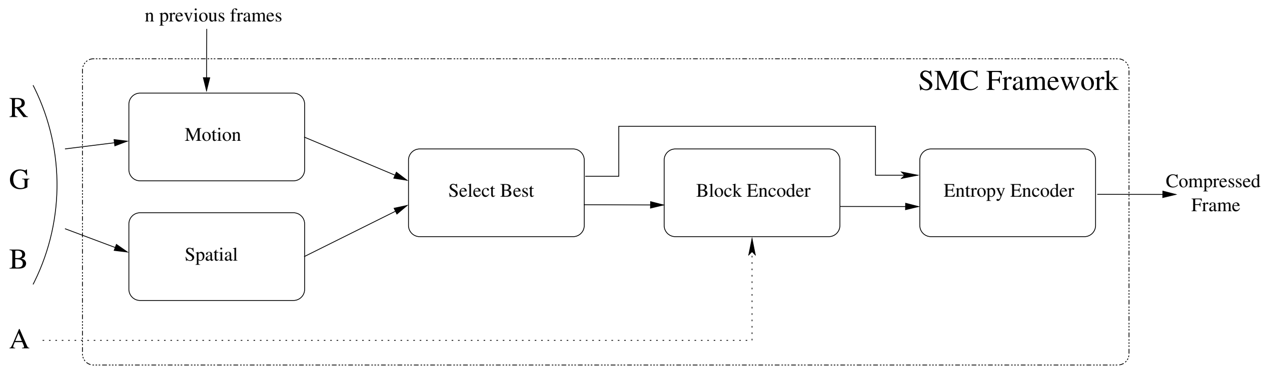

Figure 2. The individual components of the SMC compression algorithm

block matching motion estimation was chosen as the temporal compression technique. In this technique the animation frame is divided into blocks. For each block in the current frame, the most similar block is searched for in a previous frame. The criterion, to decide how similar a block is, can be to minimize the mean square error between the blocks or to minimize the absolute difference. The absolute difference is used in SMC because this is faster than the mean square error. A pointer to the best matching block is stored and is referred to as the motion vector. It consists of two components, x and y, for movement. A residual image could be obtained with the difference between the original frame and the frame built with blocks from the previous frame obtained by using the motion vectors.

The result of a block matching technique is affected by a number of parameters:

Firstly, there is a trade off between the entropy of the residual and the size of the motion vectors. A low entropy of the residual image is obtained when the size of the blocks used in block matching is small. So, a size of one pixel by one pixel produces the lowest entropy of the residual image. However, the size of the motion vectors becomes important in this case, because there is one motion vector associated with each pixel. The final compression ratio is therefore low. On the other hand, using blocks of a bigger size greatly reduces the amount of motion vectors. For example, a block of 16 by 16 pixels will produce one motion vector for every 256 pixels. In this case, the entropy of the residual image is badly affected, because it is more difficult to find exact matches when the blocks are big. The size of the blocks could be adjusted depending on the activity of different areas inside the frames.

Secondly, the size of the search space in the previous frame significantly affects the execution time and the compression ratios. A small search space is faster but produces low compression ratios. A bigger search space will produce better compression ratios, but it will be slower. The execution time increases exponentially with the search space, when the search is exhaustive. In an exhaustive search, every block within the search space is tested. Other options to exhaustive searches are logarithmic searches and multi-resolution searches. Both of them reduce the execution time to a fraction of the original, but with the incovenience that they not always provide the best matching block. Instead, they usually stop the search in a local minimum.

An analysis of experiments performed with different parameters showed that the block size does not affect the execution time. The results also showed that blocks of size two by two pixels generally produce the best compression results.

An exhaustive search method using a small search space was prefered over faster techniques because of its accuracy in low motion and static scenes. Dynamic scenes require large search spaces, which produce longer motion vectors. The entropy of these vectors is increased correspondly, and at some point the compression ratio might not necessarily become better with a bigger search space. The ACC lossless technique [5] does not implement big search spaces for similar reasons. For dynamic scenes, the small search space used in SMC is compensated by the spatial compression technique applied previously. So there is no large degradation in the resulting compression ratios.

The criterion used to choose the best matching block is the absolute difference error. Other block matching techniques include the length of the motion vector and the coherence between neighbouring vectors in the criterion. Although these techniques reduce the entropy of the motion vectors, the entropy of the residual image is increased. The residual image is very sensitive to any technique used to reduce the entropy of the vectors. For example, coherent vectors are needed to implement techniques that use matching blocks of different size. However, this will increment the entropy of the residual image. In SMC, the size of the blocks for block matching is fixed over the entire image. Lossy techniques, on the other hand, can afford the use of techniques to reduce the entropy of the vectors, because they allow the loss of information in the residual image in order to obtain good compression ratios.

3.4 Selection of the Best Compression Method

The main part in the SMC compression technique is a loop in which the best compression technique (spatial or temporal) is selected in a block by block basis. The loop starts with the spatial compression being applied to the current frame. A prediction frame is obtained as a result. This prediction frame is divided into blocks of the same size as the matching blocks - although the spatial technique was applied pixel by pixel. A vector component Z is used to specify which previous frame is used to obtain the best matching block. In the case of the spatial compression, the vector component Z is filled completely with zeros. Zero means that no previous frame is used and therefore the block is compressed with a spatial technique. The second iteration consists of applying the block matching technique using the first previous frame (i-1). The best matching blocks of the previous frame are compared with the prediction blocks obtained with the spatial technique. The best blocks are selected, that is the ones which are more similar to the original blocks in the current frame. The vector component Z will be filled with 1 only in places where the best matching block is better than the spatial predicted block. The loop is iterated for the number of reference frames (i-2, i-3, ...). At the end of this loop three vector components X, Y and Z, and a prediction frame will be obtained. The prediction frame is made by spatial predicted blocks and temporal blocks (obtain with block matching). The final residual image is obtained by subtracting the prediction frame from the current frame.

3.5 Removal of Redundant Motion Vectors

When the spatial compression technique is used, the motion vectors are not required. So, for every motion vector, where its vector component Z is equal to zero, the vector components X and Y can simply be removed. This is why spatial compression is prefered over temporal compression when both techniques produce the same absolute difference error.

3.6 Block Area Coding

The vector component Z is made of a small alphabet. Areas filled with the same value (reference frame number) are likely to occur. A good technique to reduce the redundancy of these areas is block area coding. This technique is very similar to runlength coding. The difference is that a rectangular area instead of a single line is used. Block area coding is developed in [4].

The block area coding technique consists of encoding a large area filled with the same colour by using a special symbol, followed by the size of the area and its colour. So, four symbols are required to encode a rectangular area filled with the same colour. The special symbol must be one that is not used as a colour. If a block, which is larger than the minimum size allowed, is not found then the original pixels are kept intact. The final encoded image consists of a stream of symbols that represent pixels until the special symbol for a block is found. In this case, the next three symbols will represent the width, height and colour of the constant rectangular area. Colour symbols can follow these three last symbols until a special block symbol is found again.

Notice that in SMC the block area coding technique is applied to the vector component Z and not to the RGB channels of the animation frame. The alpha channel is also encoded using the block area coding technique.

3.7 Entropy Coding

The final step in the compression consists on applying an entropy coder. There are two entropy coders available in SMC to encode the residual, the alpha channel (after block area coding) and the motion vectors (after the removal of redundant motion vectors and block area coding).

The first option is a rice coder. Rice coding is faster than arithmetic coding. It gives good compression ratios when it is applied to the compression of residuals. In SMC, this option is available for a fast compression of the residual. The motion vectors and the alpha channel are entropy coded using an arithmetic coder.

The second option consists of applying the rice coder to the residual. The result is then coded again with an arithmetic coder. Our experiments showed that this combination is faster and produces better compression ratios than just applying arithmetic coding. The motion vectors and the alpha channel are entropy coded using only the arithmetic coder.

Rice coding compression ratios are particularly good when the histogram of the residual (or any other file) has an exponential form. Residual images generated by prediction techniques and block matching techniques have approximately this form. The symbols of the residual produced in SMC are ordered, from highest frequency counts to lowest frequency counts, in order to obtain an approximation of the exponential form. This ordering procedure can also speed up the arithmetic decompression, if the arithmetic coder were applied alone, but in SMC we are always applying rice coding first.

Arithmetic coding produces compression ratios nearly optimal. This is, a compression ratio suggested by the first order entropy of the image. However, arithmetic coding is slow. We obtained a little speed up by applying rice coding first, followed by the arithmetic coding. The compression ratio was also a little better. Notice that we are using this combination only for the residual. The motion vectors and the alpha channel use arithmetic coding alone. This is because these last components do not have an exponential histogram. Their size is also much smaller than the residual, so there is no big impact on the final execution time.

3.8 Number of Previous Frames

It is recommended that the animation sequence be split up into small sub-groups. The recommended length of each sub-group is about 25 animation frames or even smaller. Each sub-group is then encoded separately. More frames in the sub-group does increase a little bit the compression but makes it more difficult to decompress a particular animation frame from the sub-group. More specifically, all the previous frames in the sub-group must be decompressed to be able to decompress a particular frame.

The number of previous frames (reference frames) used to encode a particular frame is set up by the user. Usually, this number is one or three previous frames. Notice that this number is different from that in the discussion above. Therefore, all the previous frames in a sub-group have to be decoded in order to decode a particular frame even when one or three previous frames are used to encode a frame. This is because the previous frames must be available uncompressed. So to decompress the previous frames, the previous frames to the previous frames need to be decompressed and so on.

Suppose the user selected three previous frames (reference frames) to be used by the SMC encoding technique. The first frame in a sub-group of frames does not have any previous frames. So the SMC technique applies only spatial compression as the temporal compression cannot be applied without previous frames. The second frame in the sub-group can be temporal encoded, but using a maximum of one previous frame. The third frame in the sub-group is encoded with two previous frames, and finally, the fourth and subsequent frames can be encoded with three previous frames.

Increasing the number of previous frames used to encode a frame increases the calculation time. It is suggested to keep the number of previous frames small. The previous frames (reference frames) are stored in main memory in order to perform the block matching searches. Consequently, increasing the number of previous frames requires more main memory. However, the implementation of SMC can be easily modified to keep only one previous frame in main memory at any time, even when the user selects several previous frames (reference frames) to perform the encoding.

3.9 SMC Decompression

The steps to perform the decompression correspond to the steps to perform the compression in inverse order. The entropy decoders are applied first (in inverse order). The alpha channel and the vector component Z are decoded by filling the areas that were encoded with the same colour. The redundant motion vectors are restored again. This is done because the motion vectors are misplaced without the redundant vectors. Blocks from previous frames are copied to reconstruct the current frame. The residual is added to them in order to obtain the original blocks. Finally, the spatial predictor is used together with the residual to obtain the original blocks that were spatial encoded.

3.10 SMC Compression Parameters

There are several parameters that affect the final compression ratio and the speed of the compression. These are listed below:

1. *Search space:* The size of the search space used in block matching. The execution time usually increases to the square of the search space. The compression ratio is usually better with bigger search spaces.

2. *Block size:* The size of the blocks used in block matching, and in the selection of the best technique (spatial or temporal). This is typically set to be a size of 2, which usually gives the best compression ratio. Generally, the size of the blocks does not affect the execution time.

3. *Previous frames:* The number of previous frames used in temporal compression. This is typically set to either one or three frames. The execution time usually increases linearly with the number of previous frames. The compression ratio can increase using more previous frames.

4. *Applying arithmetic coding to the residual:* This is an option. If it is false, the arithmetic coder will not be used. If it is true, the arithmetic coder will be applied. The rice encoder is always used to encode the residual, regardless of this option. Arithmetic coding is slower but produces better compression ratios.

Although the user can specify each of these parameters, a different approach is used in SMC. The user can specify the time available to do the compression and a special module can select the best parameters given this time constrain. This is explained with more details in the next sections. The advantage of this approach is that the user can use the compression software without knowledge of how the different parameters affect the compression. This also allows the compression software to produce better compression ratios when faster machines are used.

4 SMC Compression Application Steering

A use of the PACE toolkit developed at Warwick is in adding a performance model to the final execution code such that the overall execution can be optimised for a particular problem. A common case in distributed systems is a time constraint, where a problem has a limited execution time on the system. Optimising the SMC compression algorithm with the use of the Application Steering method is described in this section.

Executing the SMC compression algorithm on a particular data set, hardware system, and with a given set of compression parameters, results in a compression ratio and an execution time that is dependant on all three of these parameters. The idea of application steering, is to add a PACE performance model of the application to the overall implementation, which optimises the execution for the problem concerned. Figure 3 describes this method.

For example, it is required in this problem to use application steering to optimise the compression ratio gained for a certain time constraint. Therefore, a performance model of the compression algorithm is added to the executable, and chooses the best compression parameters to use to fit the time constraint. All the time predictions are found from exhaustive accesses to the PACE performance model executable.

Extending the compression algorithm into the application steering system shown in Figure 3 requires two distinct parts. The first is to create the PACE performance model, and then secondly to add this model to the final application. These two parts are described in detail below.

4.1 Compression Model Implementation

Before designing and implementing the performance model within PACE, it is essential to have a good knowledge of the structure of the program that you are writing the performance model for. Luckily, the full source code was available, so this was not a problem. One possible design of the model would be to have an application object which just called one big subtask which represented the whole compression algorithm, but this would result in a messy and cluttered model which would be very hard to maintain and debug. Therefore, it was decided that each main section of the algorithm would be split up into a total of five subtasks, each of which would be called by the application object when need be. These five sections would be the spatial, temporal, block, arithmetic, and rice encoding algorithms. A simple parallel template is then used to complete the model, which just describes a sequential program, as the compression algorithm model being implemented was a sequential one. The final model is represented in Figure 4.

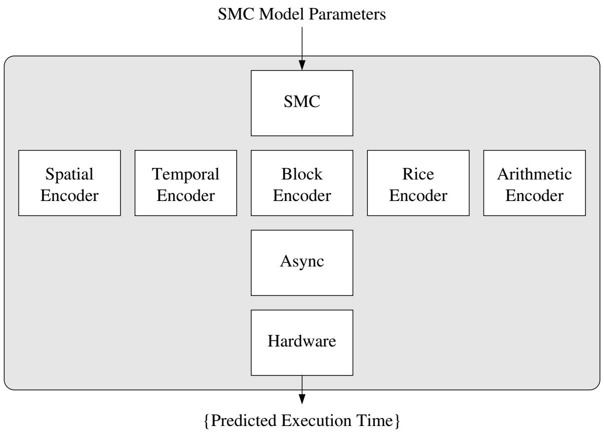

Figure 4. The final SMC PACE performance model, consiting of five sequential subtask objects, and one application, parallel template and hardware object

4.1.1 Application Object

The application object was the first to be designed, and included all the algorithm's input parameters, the subtasks for each part of the algorithm explained previously, templates for sequential processing, and hardware specifications that were to be used throughout the model.

The input parameters to the model are the algorithm parameters such as the number of frames to be compressed, the number of previous frames searched for during the temporal encoding, the width and height of each frame (assumed constant throughout the video stream), and so on. The application object also passes the required input parameters to the individual subtasks that require them. For example, the number of previous frames parameter is passed to the temporal subtask only, as this is the only part of the compression algorithm that looks at previous frames. Calls to these subtasks are then managed by the application object depending on these input parameters, and the current frame being processed. For example, if the number of channels encoded within the images includes the alpha channel as well as the normal RGB channels, then the application model has to call the block and arithmetic subtasks again for each frame, to compensate for the extra time taken to execute this functionality within the algorithm. The hardware specifications point the model to the correct hardware platform that the compression algorithm will be running on, allowing the model to predict the execution time with the highest amount of accuracy.

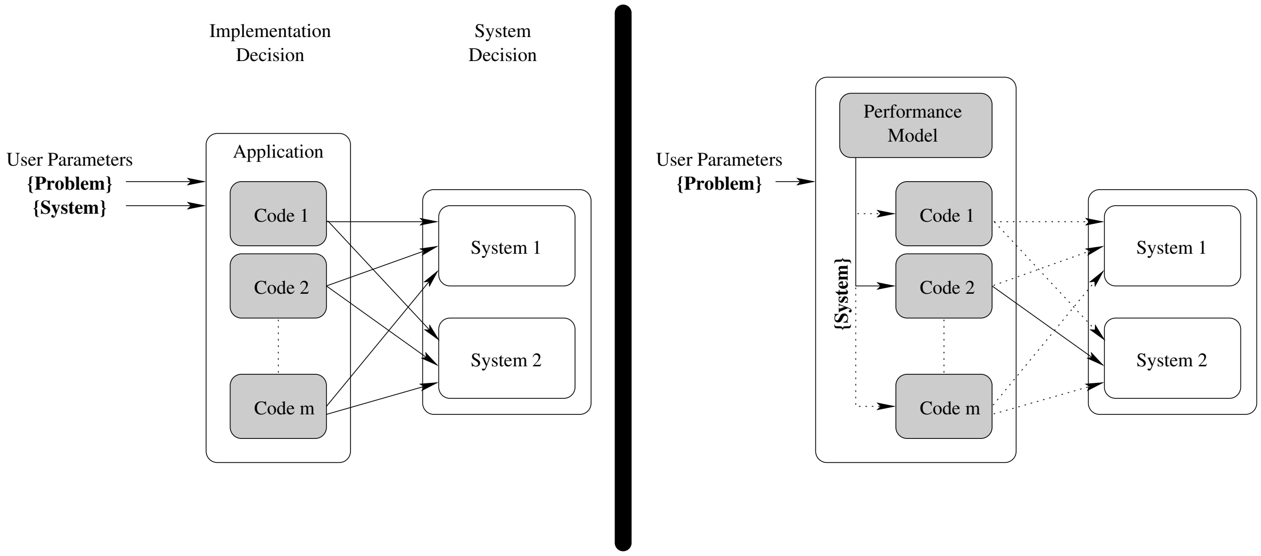

Figure 3. Application Steering Methodology

4.1.2 Subtask Objects

The design of the subtasks for the individual functionality of the algorithm is split into two distinct parts. These are porting the original code to the PACE instruction system, and organising the parameters sent by the application model within the subtask.

Porting the original code into the PACE instruction system is made simple with a PACE tool called 'capp'. 'Capp' takes C code from a file, translates it to instructions which PACE can understand, and then writes the result out to the standard output. Each original C function is returned as a PACE 'cflow' expression, which evaluates the function and predicts the execution time. In addition to this, probabilities and loop counts have to be assigned to each conditional branch within the 'cflow' expression, so that the performance model knows the likelihood for each 'if' and 'case' statements, and the number of repetitions for 'for' and 'while' statements. The rest of the subtask consists of assigning these probabilities and loop counts, which are normally linked to the parameters passed to the subtask from the application object. The 'cflow' statement to be predicted first is also assigned.

Assigning these probabilities and loop counts to the cflow statements are not as simple as it may first appear, even if a good knowledge of the original software is available. Even the author of the software can only guess these values, and although some of these guesses may be reasonably accurate, they often lead to large inaccuracies in the accuracy of the model prediction. To overcome this misfortune, a profiling tool was used to find values for these probability and loop count assignments over many repetitions of the algorithm. This profiling method is described later within this section, as it is physically used at the later stages of the model implementation.

4.1.3 Parallel Template Object

As the SMC compression algorithm was a wholly sequential one, a single, sequential parallel template was necessary. The 'async' parallel template, which accompanies PACE, was used.

4.1.4 Hardware Object

The hardware object used was for that of the hardware system used throughout the writing of this report and the gaining of the compression and application steering results found in the next section of this report. The system used was a Sun Ultra SPARC V, running at 360 MHz under the Solaris VII operating system. An accurate benchmark of this system when idle was created prior to the implementation of the model.

4.1.5 Increasing Accuracy (Profiling and Data Dependancy)

Increasing the accuracy of the model involves using profiling techniques to increase the accuracies of the probabilities and loop counts found within the subtasks, and also to compare varying data dependencies which may affect the execution performance.

Vast debugging information can be retrieved from the C compiler 'gcc' if the correct arguments to the compiler are used, and then the profiling tool 'gcov' can take this debugging output and create a list of the number of times each function, and even line of code, was executed. This information gives the implementer of the model a huge hand in vastly increasing the accuracies of the model.

Data dependencies, especially in a compression algorithm, can greatly affect the time taken to execute, providing even greater inaccuracy in the model. For example, the version of the model created from profiling with complex video was over 100% inaccurate when a simple video stream was used. It soon became apparent that the model would have to be fine-tuned so that data dependency could be taken into account.

To do this, an extension to the PACE performance model had to be written which would scan the video stream to be compressed, and make assumptions about the complexity of the video, which could then be passed to the model to refine the prediction. Running the compression algorithm on both the complex and simple streams and comparing the profiling results showed that the main differences occurred within the temporal and arithmetic sections, where the comparison between the current and previous frames where found. As explained previously, if a perfect match is found between a block in the current frame and one of the previous frames then the other previous frames will not be compared, which saves a lot of time within the algorithm. This explains why a simple video stream, where its is likely that a previous frame is very similar to the current frame, will compress in a much quicker time than a complex stream, as only a small fraction of the previous frames are being compared.

Now that the source for the data dependencies was found, the extension to the model could be designed and implemented. This extension scans the video stream and returns the probability that all the previous frames will be scanned. This probability is then passed as a parameter to the model so that the temporal and arithmetic subtasks are refined for the specific data. Adding this data dependency part to the model helped increase the accuracy of the model to predict the execution time within five percent of the actual time.

4.2 Execution Time Constraint

With a highly accurate performance model designed and implemented, it was now possible to return to the main aim of constraining the compression parameters to obtain the highest possible compression results within a certain time constraint. Another piece of software was implemented, which continuously used the model to predict the execution time for various input parameters until the best parameters to fit the time constraint were found. These parameters were then fed into the original compression algorithm.

A few examples and their results of the final performance model can be found within the next section.

5 Results

The total time to execute the compression is affected by a number of different factors. On one side we have the characteristics of the machine used to run the compression and on the other side we have the characteristics of the data. So, given the compression parameters, the characteristics of the machine and the characteristics of the data, we can produce a good prediction of the total execution time.

In the case of SMC lossless compression, a fast preprocessing module to scan the animation data before the compression was developed in order to obtain an approximated measurement of the temporal redundancies in the data. This measurement or probability, which is part of the data characteristics, is then passed to PACE. This produces more accurate predictions.

The results obtained from the performance model are split into two categories. The first focuses just on the performance model, and compares the predicted times verses the actual compression times for a variety of input parameters and for both the Drunky and Limbo video streams. Twenty-five frames were selected, and at various parts of the video stream, to allow the accuracy of the data dependency extension to the model to become apparent. These results show just how accurate the final implementation of the model, including the data dependency predictions, were.

The second part of these results focuses on the final system, where the compression result is constrained to a specific value of time. Each graph shows both the predicted time to compress twenty frames and the actual real compression time for the parameters chosen by the model. Tables are also provided to complement the graphs, which show the compression parameters predicted by the model to fit the time constraint.

5.1 Performance Model Prediction

The graphical results shown below compare the performance model's predicted time with the actual time taken to compress the described frames and compression level. Each compression level defined within the compression algorithm is a unique set of input parameters to the compression, which allow for better compression results to be achieved, although within a longer space of time.

Different areas of the Drunky video stream were used to show the accuracy of the data dependency part of the model. Frames 360 to 384 are very different from each other as the camera is moving quickly, and this results in a longer time to search for similar blocks in the previous frames, and therefore a longer time for the compression to complete. Frames 850 to 874, on the other hand, are very similar, and this results in a much shorter time to search for similar blocks and therefore a much shorter time to com-

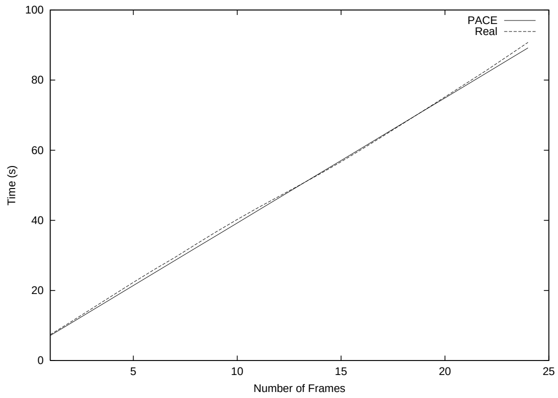

Figure 5. Compression Time Prediction for DRUNKY (Compression Level 1, Frames 360-384)

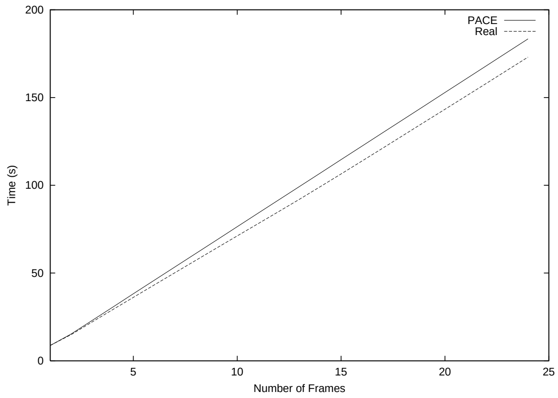

Figure 6. Compression Time Prediction for DRUNKY (Compression Level 4, Frames 850-874)

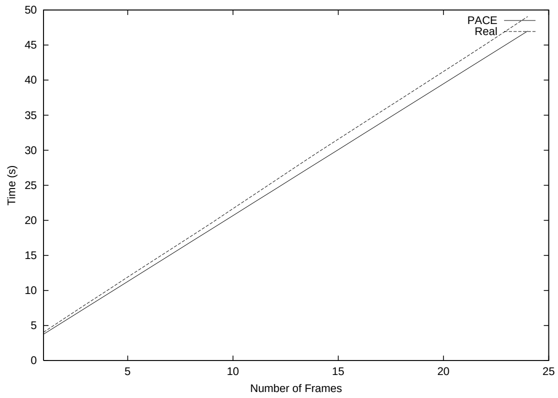

Figure 7. Compression Time Prediction for LIMBO (Compression Level 1, Frames 37-61)

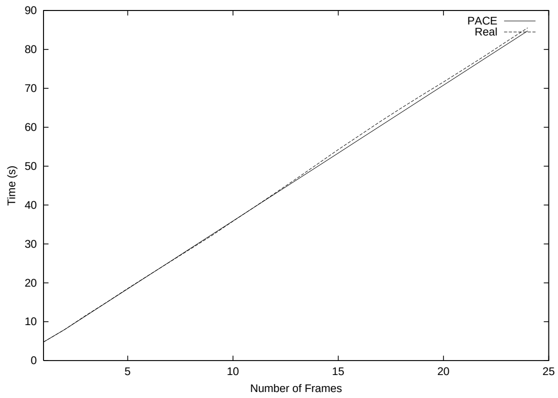

Figure 8. Compression Time Time Prediction for LIMBO (Compression Level 4, Frames 37-61)

plete the compression. It is also noted that the compression ratio gained is also considerably higher for the latter set of frames. These two areas of the Drunky video sequence were chosen on purpose to show that the data dependency part of the model does in fact tune the model to correctly predict the execution time for varying data sets.

Figures 5, 6, 7, and 8 support the performance model prediction section of the results achieved. The percentage error between the predicted and real compression times is less than 8% for all the results shown.

5.2 Time Constraint Prediction

Tables 1, 2 and 3 show the results of the time constraint extension to the performance model. For various parts of the Drunky video stream, the compression of twenty frames within time constraints of between 50 and 250 seconds were predicted using the performance model. The predicted compression parameters to constrain the compression to these times were then input to the compression algorithm, and the time taken to compress with these parameters was also recorded.

6 Conclusion

Within this paper it has been shown how an application steering method can be used to optimise applications for distributed systems where certain constraints have to be met. The example used here was of a lossless compression technique, where a time constraint may be enforced for a particular system where a large number of applications have to be scheduled. The results show that the compression parameters chosen by the performance model optimise the compression technique to gain the best possible compression ratio as the time constraint becomes more relaxed.

The application steering method involves adding the

PACE performance model with the compression executable such that the compression parameters are chosen by the PACE model instead of the user. Thus the user chooses the time constraint, which the PACE model takes and calculates the compression parameters which will results in the highest compression ratio.

The PACE system is currently being extended to allow for the constant changes that are likely within distributed systems such as the Computational GRIDS.

7 Acknowledgment

This work is funded in part by DARPA contract N66001-97-C-8530, awarded under the Performance Technology Initiative administered by NOSC.

The Drunky video stream that were referred to throughout this report were donated by Mental Images. The Limbo video stream was acquired from Blue Moon Rendering Tools (BMRT).

References

[1] A.M. Alkindi, D.J. Kerbyson, E. Papaefstathiou, G.R. Nudd, "Run-time Optimisation Using Dynamic Performance Prediction".

[2] L. DeRose, Y. Zhang, D.A. Reed, "SvPablo: A Multi-Language Performance Analysis System", Proceedings of the 10th International Conference on Computer Performance. Spain. 252-255 (1998).

[3] I. Foster, C. Kesselman, "The Grid : Blueprint for a New Computing Infrastructure", Morgan Kaufmann. 279-290 (1998).

[4] J.M. Gilbert, R.W. Brodersen, "A Lossless 2-D Image Compression Technique for Synthetic Discrete-Tone Images", IEEE DCC. 359-368 (1998).

[5] E. Groller, W. Stockers, "ACC - Lossless Data Compression of Animation Sequences", IFIP Transactions of Graphics, Design and Visualization. 75 (1993).

[6] W. Gu, G. Eisenhauer, K. Schwan, "On-line Monitoring and Steering of Parallel Programs", Concurrency: Practice and Experience. **10(9)**, 699-736 (1998).

[7] B.P. Miller, M.D. Callaghan, J.M. Cargille, J.K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam, T. Newhall, "The Paradyn Parallel Performance Measurement Tools", IEEE Computer. **28(11)** 27-46 (1995).

[8] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, D.V. Wilcox, "PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems", International Journal of High Performance Computing Applications, Special Issues on Performance Modelling. **14(3)**, 228-251 (2000).

[9] E. Papaefstathiou, D.J. Kerbyson, G.R. Nudd, T.J. Atherton, "An Overview of the CHIP^{3}S Performance Prediction Toolset for Parallel Systems", 8th ISCA Int. Conf on Parallel and Distributed Computing Systems. Florida USA. 527-533 (1995).

[10] M. Weinberger, G. Seroussi, G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS", Hewlett Packard Labs HPL. 98-193 (1998).

[11] X. Wu, N. Memon, "Context-based Adaptive Lossless Image Coding", IEEE Transactions. **45(4)** 437-444 (1997).

[12] H.C. Yun, B.K. Guenter, R.M. Mersereau, "Lossless Compression of Computer-Generated Animation Frames", ACM Transactions on Graphics. **16(4)** 359-396 (1997).

Time Constraint	Time Prediction	Real Time Taken	Compression Ratio	No. Previous Frames	Search Space	Arithmetic Encoding Used
50.00	62.74	66.16	4.67	0	1	OFF
60.00	62.74	66.34	4.67	0	1	OFF
70.00	69.38	89.26	4.92	0	1	ON
80.00	69.38	89.15	4.92	0	1	ON
90.00	85.89	92.77	5.20	1	1	OFF
100.00	92.54	111.06	5.87	1	1	ON
110.00	92.54	110.83	5.87	1	1	ON
120.00	92.54	111.14	5.87	1	1	ON
130.00	92.54	111.02	5.87	1	1	ON
140.00	92.54	110.89	5.87	1	1	ON
150.00	92.54	111.09	5.87	1	1	ON
160.00	92.54	110.86	5.87	1	1	ON
170.00	92.54	110.89	5.87	1	1	ON
180.00	92.54	111.09	5.87	1	1	ON
190.00	182.47	181.41	6.06	3	2	ON
200.00	182.47	181.52	6.06	3	2	ON
210.00	182.47	181.42	6.06	3	2	ON
220.00	210.84	201.99	6.07	4	2	ON
230.00	210.84	202.19	6.07	4	2	ON
240.00	237.44	220.86	6.07	5	2	ON
250.00	237.44	220.65	6.07	5	2	ON

Table 1. Time contraint predictions and compression ratio achieved for DRUNKY: Frames 420-429

Time Constraint	Time Prediction	Real Time Taken	Compression Ratio	No. Previous Frames	Search Space	Arithmetic Encoding Used
80.00	69.92	88.49	5.10	0	1	ON
90.00	85.12	92.96	5.31	1	1	OFF
100.00	91.61	110.35	6.06	1	1	ON
110.00	91.61	110.19	6.06	1	1	ON
120.00	91.61	110.04	6.06	1	1	ON
130.00	91.61	110.34	6.06	1	1	ON
140.00	91.61	110.15	6.06	1	1	ON
150.00	91.61	110.20	6.06	1	1	ON
160.00	91.61	110.27	6.06	1	1	ON
170.00	91.61	110.19	6.06	1	1	ON
180.00	173.86	177.73	6.34	3	2	ON
190.00	173.86	177.76	6.34	3	2	ON
200.00	199.88	197.67	6.35	4	2	ON
210.00	199.88	197.61	6.35	4	2	ON
220.00	199.88	197.52	6.35	4	2	ON
230.00	224.28	215.66	6.37	5	2	ON
240.00	224.28	215.80	6.37	5	2	ON
250.00	247.05	232.79	6.38	6	2	ON

Table 2. Time contraint predictions and compression ratio achieved for DRUNKY: Frames 670-679

Time Constraint	Time Prediction	Real Time Taken	Compression Ratio	No. Previous Frames	Search Space	Arithmetic Encoding Used
50.00	61.27	63.39	5.42	0	1	OFF
60.00	61.27	62.97	5.42	0	1	OFF
70.00	67.18	82.13	5.86	0	1	ON
80.00	67.18	82.12	5.86	0	1	ON
90.00	88.07	102.45	7.35	1	1	ON
100.00	88.07	102.32	7.35	1	1	ON
110.00	88.07	102.58	7.35	1	1	ON
120.00	88.07	102.46	7.35	1	1	ON
130.00	88.07	102.63	7.35	1	1	ON
140.00	88.07	102.34	7.35	1	1	ON
150.00	144.07	156.21	7.78	3	2	ON
160.00	144.07	156.32	7.78	3	2	ON
170.00	162.03	171.90	7.81	4	2	ON
180.00	178.87	186.10	7.82	5	2	ON
190.00	178.87	186.09	7.82	5	2	ON
200.00	194.59	199.24	7.83	6	2	ON
210.00	209.18	211.38	7.84	7	2	ON
220.00	209.18	211.54	7.84	7	2	ON
230.00	222.65	222.57	7.84	8	2	ON
240.00	238.37	246.97	7.84	3	3	ON
250.00	238.37	247.05	7.84	3	3	ON

Table 3. Time contraint predictions and compression ratio achieved for DRUNKY: Frames 850-859