Accelerated Simulation of Communication Waveforms Using The MathWorks Parallel Computing Toolbox

Brendan Garvey
(General Dynamics C4 Systems, Scottsdale, AZ, USA)
brendan.garvey@gdc4s.com

Updated 3-5-2008

Outline

- Introduction
- Overview of The MathWorks Parallel Computing Toolbox
- Key Concept
- Requirements / Features for a Parallel Monte Carlo Waveform Simulation
- Implementation Details
- Waveform Models & Performance Results
- Cost Benefit Analysis
- Summary & Conclusions

Introduction (1)

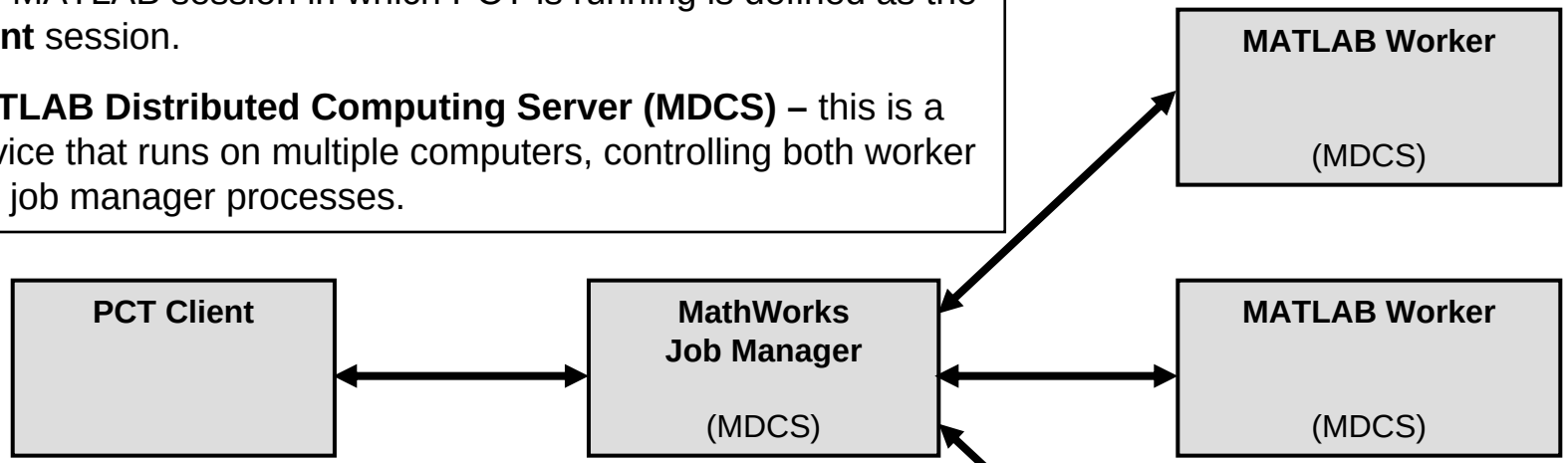
- Developing new communication algorithms and waveforms typically requires significant engineering effort and extensive, time consuming simulations. Although MATLAB and Simulink have proven valuable for this type of work, designs of even modest complexity can require long simulation times.
- The Mathworks now offers a parallel computing solution, the Parallel Computing Toolbox (**PCT**), which has the potential to address this problem.
- This presentation introduces a methodology and algorithm for significantly accelerating the simulation of communication waveforms using the Parallel Computing Toolbox.

Introduction (2)

- In the approach proposed here, a single Monte Carlo simulation is broken into numerous smaller (shorter) Monte Carlo simulations, running on multiple computers, and the results averaged together to create the final answer.
- This parallel Monte Carlo (PMC) approach is already being used by researchers and engineers in other technology areas [1, 2].
- A MATLAB m-file, **dc_sim.m**, has been developed to validate this approach and is described in this paper. The approach can be used for both MATLAB (script-based) and Simulink (model-based) simulations.
- A MATLAB script which models a forward channel in the IS-95 system has been modified to work with dc_sim(). This model was simulated on a cluster of 10 computers in order to characterize the proposed approach. A CDMA2k Simulink model is currently being modified to confirm that the approach also works with Simulink.

Overview of PCT
Basic MATLAB Parallel Computing Environment

- There are two separate products that make up The Mathworks parallel computing environment:
- **Parallel Computing Toolbox (PCT) -** this is a set of commands and functions that are executed from the MATLAB command window, or from an m-file, just like other toolboxes. The MATLAB session in which PCT is running is defined as the **client** session.
- **MATLAB Distributed Computing Server (MDCS) –** this is a service that runs on multiple computers, controlling both worker and job manager processes.



Typical client commands:

```
job = creatJob(jm1);
fhan = @IS95_wf_eval;
createTask(job, fhan, 2, {…});
submit(job);
```

Figure 1

Key Concept (1)

- A Monte Carlo simulation can be broken into smaller (shorter) Monte Carlo simulations, running in parallel, and the results averaged together to create the final estimate. This is usually referred to as “parallel Monte Carlo” (no surprise here).

- For example, if a simulation requires 1e7 symbols in order to form an adequate BER estimate, the simulation could be run on 10 workers, with each worker simulating 1e6 symbols. The results are combined to form the final BER estimate.

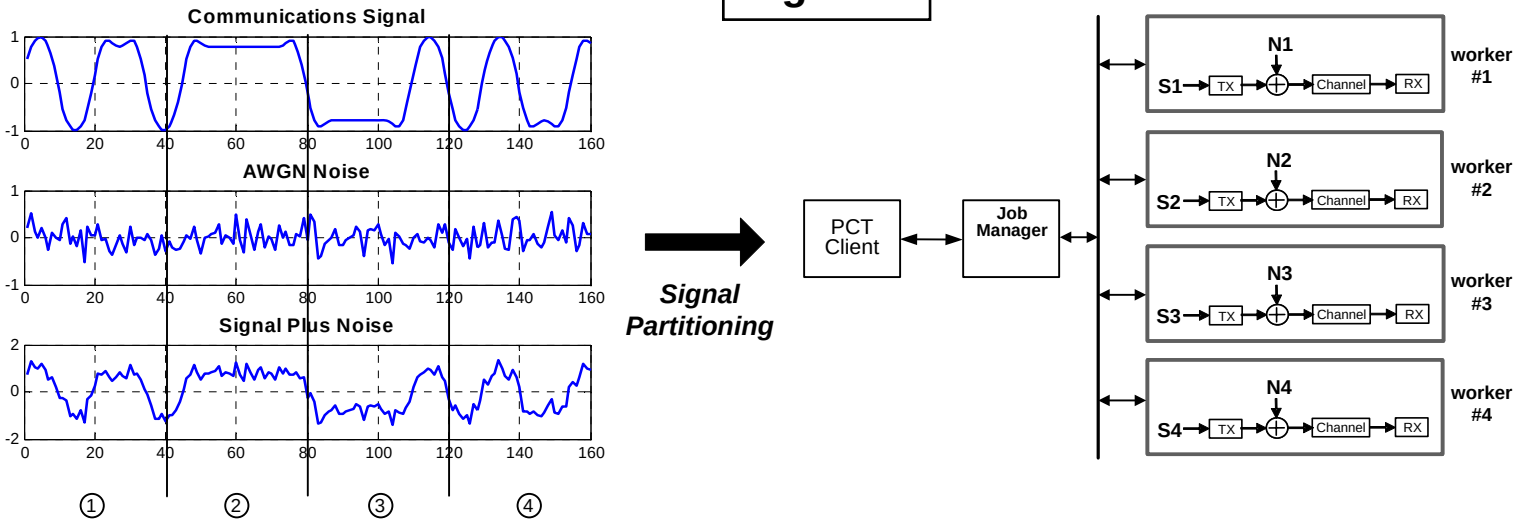
- Although this is conceptually straightforward, ***care must be taken to ensure that the individual simulations are statistically independent***.

- In the approach proposed here, function **dc_sim()** takes in a simulation at a single Eb/No value, and partitions it into multiple shorter simulations. The shorter simulations are submitted as jobs onto the workers. Function **dc_sim()** combines the results from the worker computers, and returns the combined result to the calling script.

Key Concept (2)
Assigning Simulations to Workers

- The input signal is divided into numerous simulation blocks.
- Each worker evaluates the simulation blocks assigned to it by the job manager.
- The only difference between simulations on individual workers is the noise sequence and the input data. Function dc_sim(), running on the PCT client, collects the individual worker results, and returns the results to the top-level script.
- The top-level script calculates the final BER or SER estimate.

Figure 2



Requirements / Desired Features (1)

- Now that the key concept has been presented, this section presents general requirements and desirable features for a distributed waveform simulation using a parallel Monte Carlo (PMC) approach.

- **Parallel Random Number streams** – "Parallel random number generators should … [4]
  - have no correlations between the sequences on different processors,
  - produce the same sequence for different numbers of processors,
  - and not require any data communications between processors."

    In addition, it is desirable that the random number generation method build upon existing MATLAB functionality.

    Function dc_sim() meets these requirements by using "sequence-splitting," centrally controlling the noise state table at the PCT client, and using MATLAB's randn() function.

Requirements / Desired Features (2)

- **Repeatability** – The simulation should produce identical results regardless of the number of computers used in the cluster. This requirement is also met by centrally controlling the noise seeds at the PCT client.

- **Reliability** – As the number of computers goes up, the probability of having a computer problem also goes up. The simulation should provide a means to recover from a worker failure and remove the worker from the computer cluster. This functionality has not yet been added to dc_sim().

- **Expandability** – A PMC waveform simulation should provide a method of parallelization that is independent of the waveform model, so that computing resources can be easily added or subtracted from the simulation. Function dc_sim() has been designed to meet this requirement.

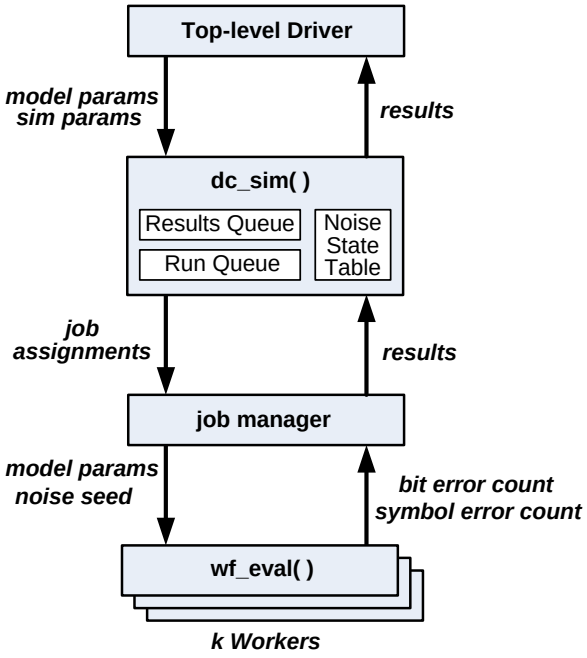
Implementation Details (1)
Introduction

- This section explains how a PMC waveform simulation can be implemented. Function dc_sim() and the modified IS-95 MATLAB model are used as an example.
- Figures 3 and 4 (a few slides ahead) show the three main components of the proposed approach – a top-level MATLAB script and two supporting functions, each which reside in its own m-file.
  - Figure 3 is a high-level block diagram of the overall simulation architecture, showing how these three components interact.
  - Figure 4 is an outline of the 3 m-files using MATLAB syntax. This is essentially pseudo-code which, for the sake of explanation, leaves out much of the detail.
- We'll first give an overview of each component, and then we'll go back and fill in the details. The approach shown here is applicable to both MATLAB simulations and Simulink models.

Implementation Details (2)
Block Diagram of Simulation Architecture

- **Top-Level Driver** – For each Eb/No value to be evaluated, the top-level driver calls dc_sim(). The top-level driver calculates the final BER or SER based on the results returned from dc_sim().
- **Function dc_sim()** – dc_sim() monitors its internal run queue and strives to keep the run queue loaded with ***k*** jobs (each job represents a simulation block that needs to be simulated). dc_sim() continues to submit jobs and record results in the results queue until the target end conditions have been met. It then returns the accumulated error counts to the top-level driver.
- **MathWorks Job Manager** – This process is part of the PCT software. It controls the assignment of jobs to workers.
- **Function wf_eval()** – This function evaluates the waveform for a single simulation block, which consists of ***L*** packets. wf_eval() returns bit and symbol error counts back to dc_sim() via the job manager.

Figure 3



Implementation Details (3)

Figure 4

Top-level Driver

```
%% Top-Level Script
.
.
.
%% Initalize dc_sim
dc_sim();

%% Main simulation loop
for i=1:length(EbNo)
    .
    .
    .
    res = dc_sim(EbNo);
end
3
```

1 — `dc_sim();`

3 — `end`

Runs on client machine

dc_sim( )

```
function res = dc_sim( )
if (init==1)
  %% initialize state table
  ...
end
if (init==0)
  while (done_jobs< job_cnt
    || done==0)
    %% sub-loop #1%%%%%%%%%
    %% check for results %%
    for i=1:qlen
      ...
    end
    %% sub-loop #2 %%%%%%%%
    %% submit jobs %%%%%%%%
    while(...)
      job = createJob(jm1);
      tsk = createTask(job, ...);
      submit(job);
    end
  end
end
```

2 — `if (init==1)`

4 — `while (done_jobs< job_cnt || done==0)`

5 — `for i=1:qlen`

6 — `while(...)`

Runs on client machine

wf_eval( )

```
function [ ] = wf_eval(I )

 x = Modulator( );
randn('state', ran_state);
noise = scale*(randn(size(x)) +
      j*randn(size(x)));
r = x+noise;
RxSD = Demodulator( r);
```

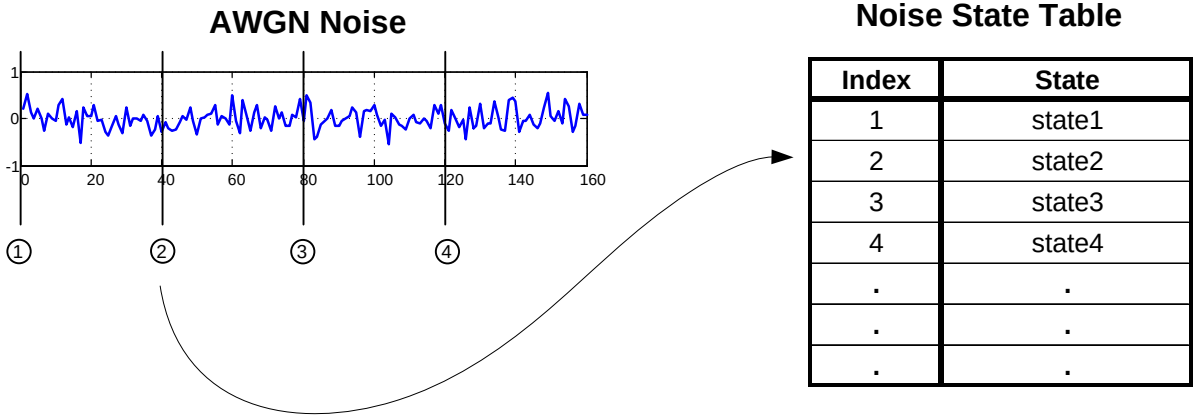
7 — `function [ ] = wf_eval(I )`

Runs on worker

Implementation Details (4)
The Noise State Table

- **"Sequence splitting" is used to create uncorrelated noise sub-sequences.**
- **Sequence splitting means taking a single sequence and dividing it up into non-overlapping sub-sequences.**
- **The noise state table holds an array of states, each state represents the start of a noise sub-sequence.**
- **Each noise sub-sequence contains the random elements needed for 1 simulation block.**

Figure 5



Implementation Details (5)
The Noise State Table

- Sequence splitting can be accomplished in a straightforward manner using the MATLAB randn() function. The MATLAB command

```
state1 = randn('state');
```

returns the current internal state of the randn() generator. Later, the randn() generator can be re-initialized to state1 using the command

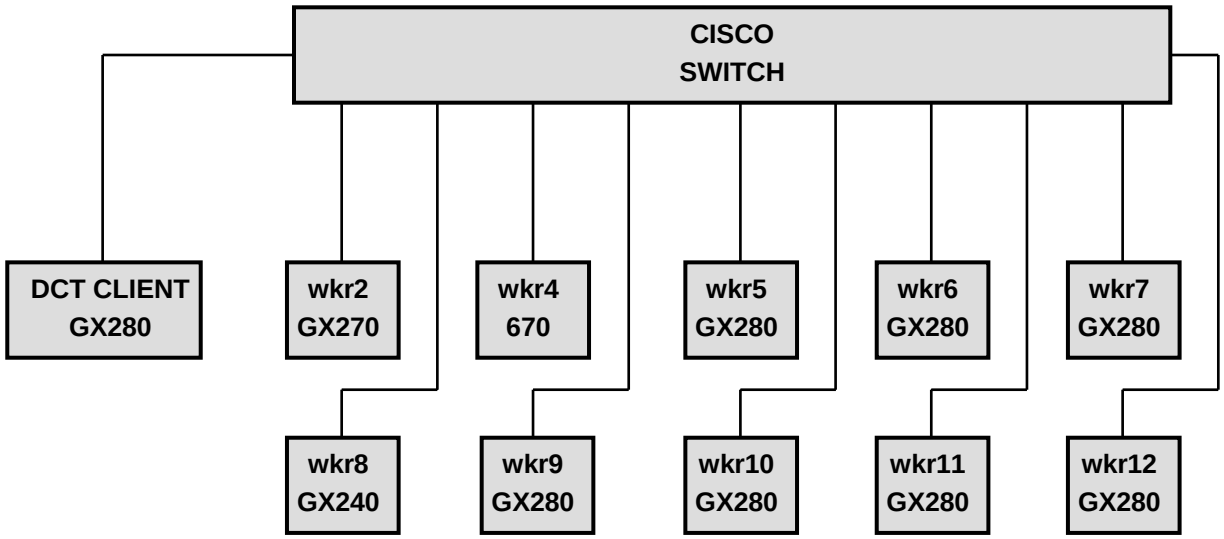
```
randn('state', state1);
```

- To generate the noise state table, we create a loop which generates M random numbers at a time (the requirements for one simulation block). After each block of M random numbers have been generated, we throw away the generated numbers, but save the state (the noise seed) into the noise state table. The noise seed is used by the worker to initialize the state of its own randn() generator.

Implementation Details (6)
Network Configuration

- The block diagram below shows the network configuration of the General Dynamics C4 Systems Computer Cluster.
- All computers are on the same sub-net.
- This is the configuration for the MATLAB model. For the Simulink simulation, wkr8 was taken out of the cluster.

Figure 6



MATLAB Model (1)
Overview

- The MATLAB model was downloaded from the MATLAB central file exchange. The original code was written by Gennady Zilberman of Ben-Gurion University in Israel. It has been modified to work with dc_sim().

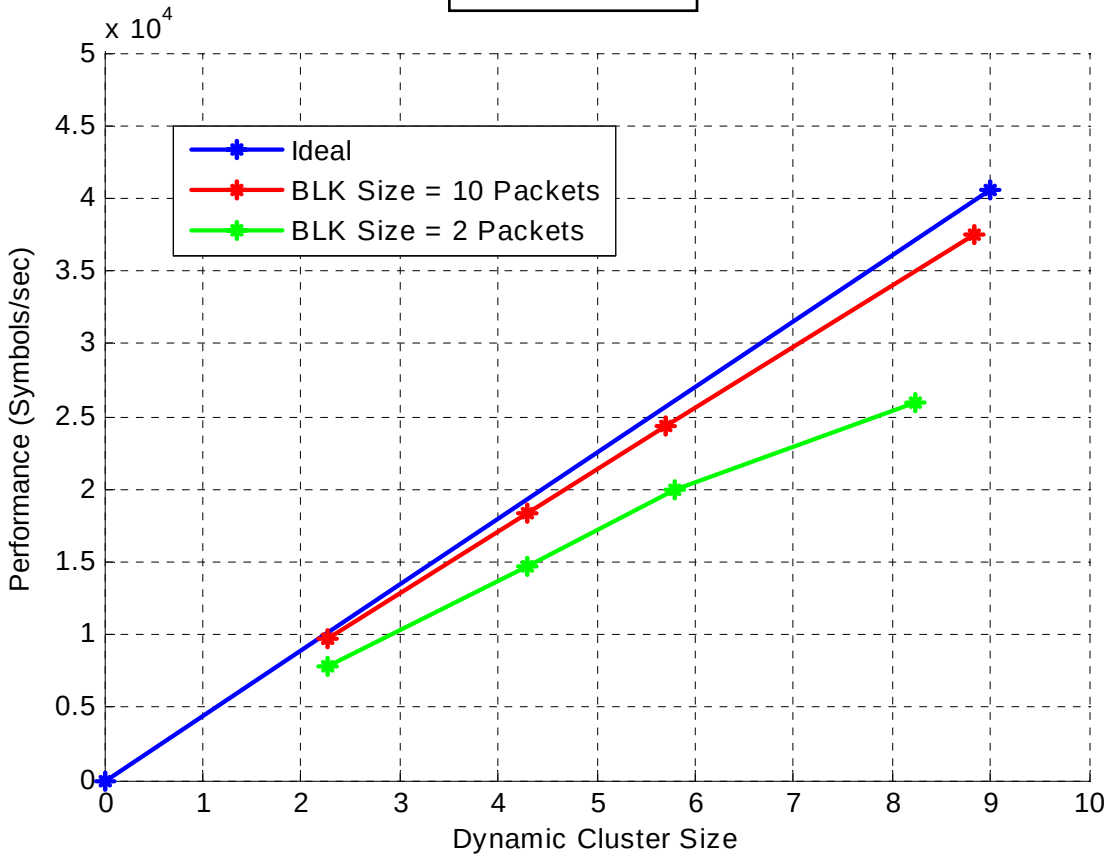
- This script models a forward channel in the IS-95 system. The block diagram is very similar to the Simulink model, see figure 8.

- Model specifics:
  - **Encoder Block –** CRC, R=1/2 K=9 Encoder, Interleaver
  - **Transmitter Block –** long code scrambling, signal point mapping, baseband filtering
  - **Channel Block** – AWGN noise
  - **Receiver Block** – baseband filtering, symbol demapping, long code descrambling
  - **Decoder Block**– Deinterleaver, Viterbi decoder (either soft or hard decision)

MATLAB Model (2)
Cluster Performance Results

- **As the simulation block size increases, the efficiency of the cluster improves.**
- **If the block size is large enough, the performance is fairly close to ideal.**

Figure 7



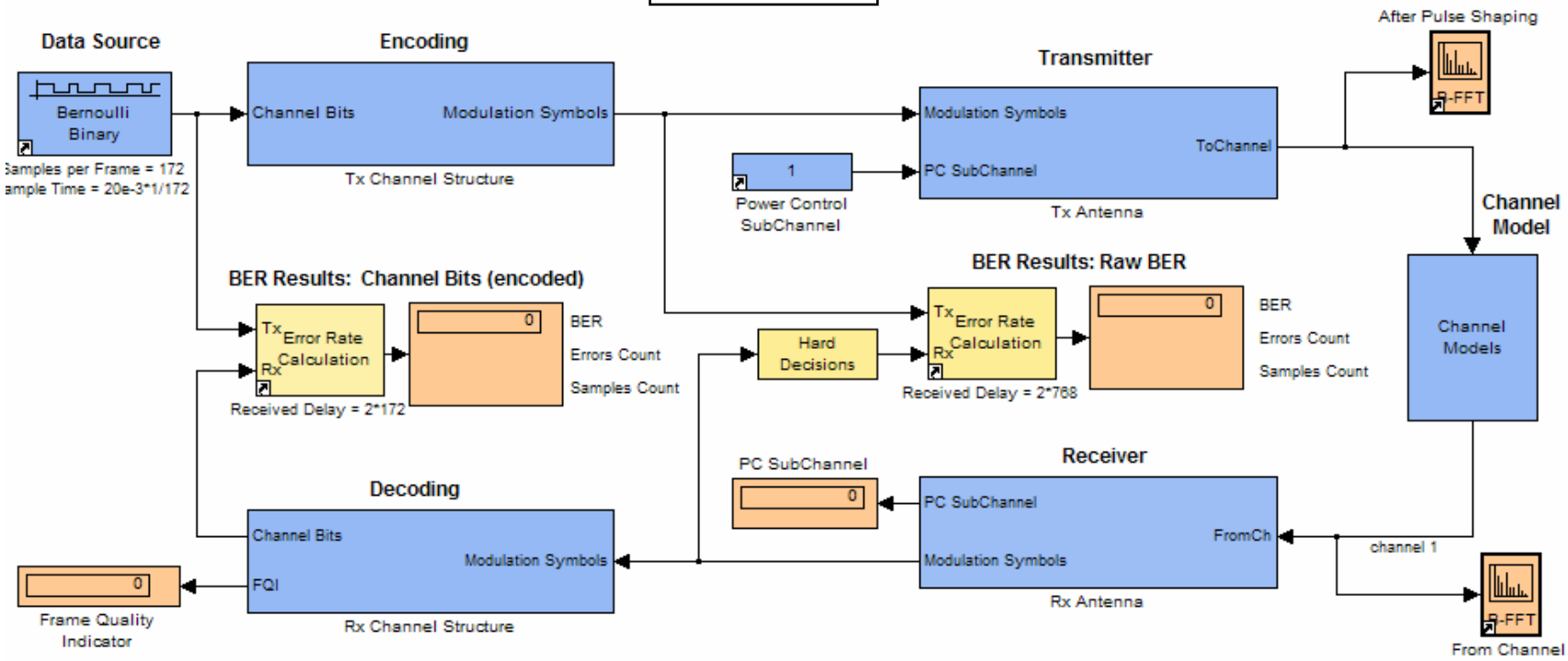
Simulink Model (1)
Simulink Model Overview

- The Simulink model is based on a demo that is part of the Simulink online documentation (see next slide). Overview of Simulink model:
- **Encoder Block –** CRC, R=1/4 Encoder, Interleaver
- **Transmitter Block –** long code scrambling, signal point mapping, baseband filtering
- **Channel Block** – Multipath fading and AWGN noise
- **Receiver Block** – baseband filtering, rake receiver, symbol demapping, long code descrambling
- **Decoder Block** – Deinterleaver, depuncture, Viterbi decoder

Simulink Model (2)
Original Simulink Model

This model is currently being modified to work with dc_sim()

Figure 8



Cost Benefit Analysis

- For 3 recent projects within our company, this is an estimate of the hours that would have been saved if a computer cluster had been used for the waveform simulations.
- Discussions with several project leaders and system engineers resulted in an estimate of 2.5% for the reduction in the overall system engineering hours required to develop a waveform.
- A computer cluster of 16 resulting in a 10x to 15x speedup was assumed.

Project	Total SYS ENG HRS for WF Development	SYS ENG HRS Saved
A	2352	59
B	3696	92
C	2350	59

Summary & Conclusions

- This paper has presented an approach for significantly accelerating the simulation of communication waveforms using the Parallel Computing Toolbox. It has been shown that, for reasonable simulation block sizes, a parallel Monte Carlo waveform simulation efficiently utilizes computer resources.
- Performance of the cluster was verified using a MATLAB model of an IS-95 forward link channel. A CDMA2k Simulink model is currently being modified to confirm that the proposed approach also works with Simulink models.
- Work has mainly focused on creation and implementation of the PMC algorithm; further work is needed to validate the simulation results of the modified models.

References

[1] J.S Kim, S.J. Byun, “A Parallel Monte Carlo Simulation on Cluster Systems for Financial Derivatives Pricing”, *The 2005 IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1040-1044, 2-5 Sept. 2005.

[2] Y.K. Dewaraja et al., “A Parallel Monte Carlo for Planar and SPECT Imaging …”, *The 2005 IEEE Nuclear Science Symposium Conference Record*, Vol. 3, 15-20 Oct. 2005.

[3] *Distributed Computing Toolbox User’s Guide*, Version 2. The Mathworks.

[4] P.D. Coddington, “Random Number Generators for Parallel Computers”, Syracuse University, pp. 2, April 28 1997.

Supplemental Notes (1)

Evaluation Methodology – Calculating Ideal Performance

- These next few slides explain the methodology used to evaluate the performance of the cluster. First, let's look at the calculation of the ideal performance:
- Assume a cluster of ***k*** workers. Note that the PCT client is not included in the performance calculation, only the workers.
- Measure the performance of each worker individually, P_i, in chips/sec.
- Sum the performance (chips/sec) of all ***k*** workers, call this P_T.
- The ideal performance is a line from (k,P_T) through the origin.

Supplemental Notes (2)

Evaluation Methodology – Calibrating the MATLAB Cluster

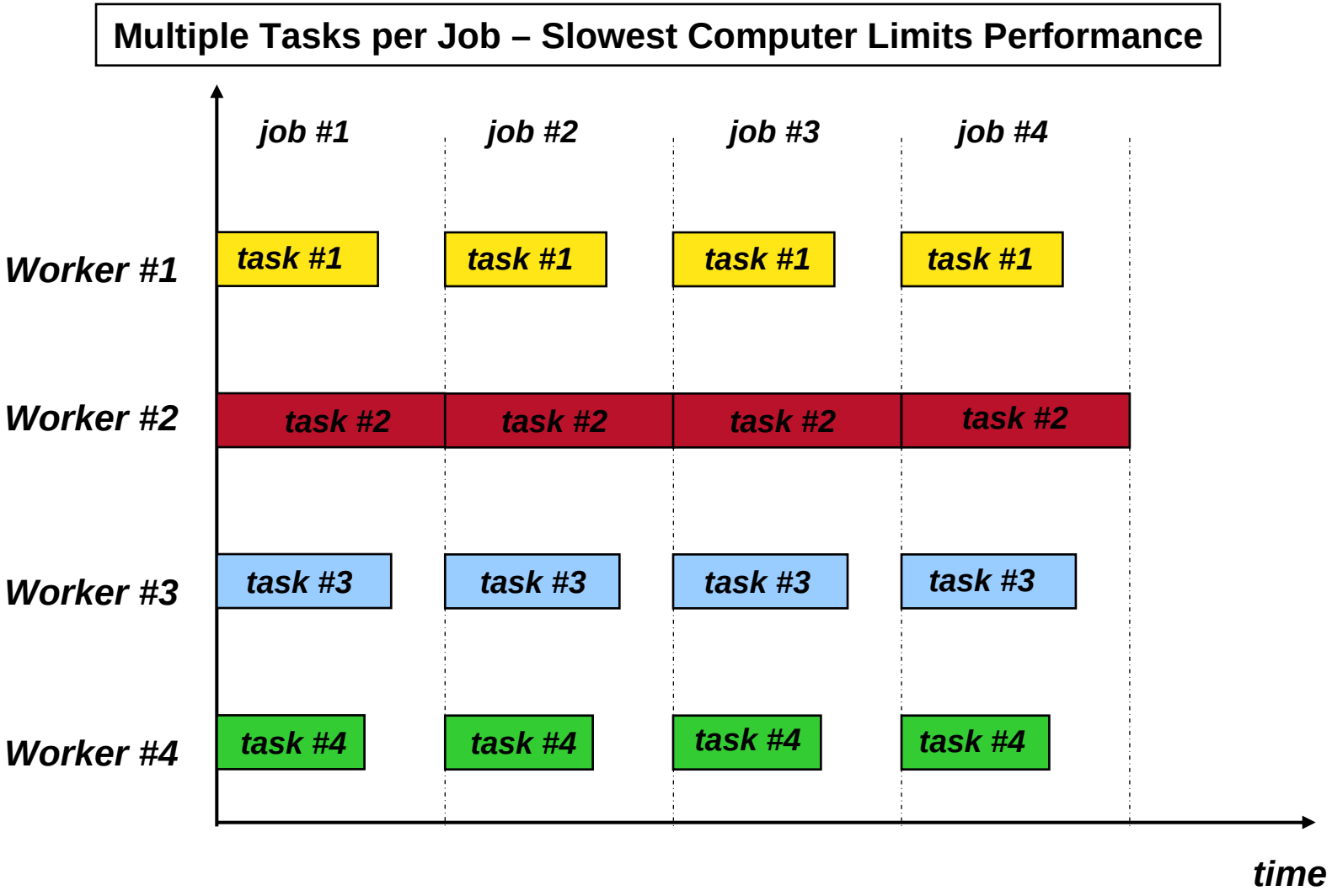
- Ideally, in order to characterize the cluster, each computer would have identical performance.
- Of course, this is not the case. A methodology was developed to scale the individual computer sizes in order to account for the difference in performance between computers.
- For a cluster of size ***k***, the computer size S of each computer (worker) is defined to be
- $S_i = (P_i/P_T)k$.
- For example, if there are 10 computers in the cluster, and the ith computer has exactly 1/10 of the performance of the cluster, then its “size” would be 1.

Supplemental Notes (3)

Evaluation Methodology – Calculation of Actual Performance

- Each point on the performance graph is calculated as follows:
- First, a target number of workers is specified. This is the maximum number of workers that dc_sim() is allowed to use during the simulation.
- During the simulation, the total computer size (the sum of the individual computers sizes) of the cluster is periodically recorded. Note that the total computer size is different than the target number of workers:
  - Each worker has a different computer size.
  - The cluster will not always be using the maximum number of allowed workers. For example, towards the end of a simulation, the number of workers will start to decrease.
- At the end of the simulation, the (x,y) performance point is defined as:
  - X: this is the average computer size (dynamic cluster size) used for the simulation.
  - Y: this is the total number of chips simulated, divided by the total time for the simulation (Y is the performance).

Supplemental Notes (4)
Multiple Tasks per Job



Supplemental Notes (5)
Single Task per Job

Single Task per Job – Overall, cluster operates more efficiently