Gedae Portability: From Simulation to DSPs to the Cell Broadband Engine

James Steed (jsteed@gedae.com), William Lundgren (wlundgren@gedae.com), Kerry Barnes (kbarnes@gedae.com)
Gedae, Inc., 1247 N. Church St., Suite 5, Moorestown, NJ 08057

Introduction

This poster is a study on the portability of applications developed in Gedae by analyzing the work that was required to move an example application from simulation on a PC to running on a DSP board (the Mercury AdapDev™ system), and then to running on a multicore processor (the Sony/IBM/Toshiba Cell Broadband Engine™ (BE)). We will illustrate how the architecture considerations were taken into account when porting the application to each system and quantify both the work required to port the application and the performance of the application on each system. The poster presentation will also include a demonstration of the application running on the Cell/BE in the Sony Playstation 3 with realtime input and output, as well as a demonstration of the simulation of the application.

Application

The example application involves tracking a model train as it goes in a circular path around its track. The application uses input audio data from four microphones placed in a circle around the track to locate the train in the audio field. Using this location, the application pans and tilts a camera to point at the engine of the train. An illustration of this environment is shown in Figure 1.

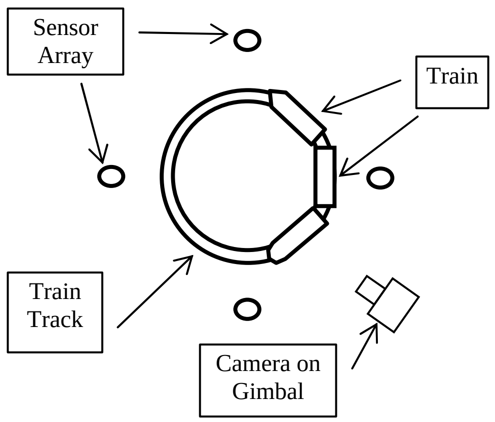


Figure 1 – The tracking algorithm targets a train running on a track, with 4 microphones as sensor inputs.

The algorithm is based on RADAR technology. A beamformer correlates a linear array of RADAR sensors to identify a target based on a beam of high correlation. In this application, the array is circular, so the high intensity in the correlation of the four channels forms a spot, as shown in Figure 2. After the spot formation forms this audio map, a detection algorithm identifies the high intensity peak corresponding to the train, and pan and tilt angles are computed to reposition the camera.

Because this application must continue to work in noisy environments, several approaches are used to reduce jitter and ensure smooth tracking. The input channels are run through low pass filtering to remove frequencies outside the desired band. In the detection algorithm, several peaks are identified and tested, and feedback is used to monitor the speed and direction of the train to help rule out spurious peaks in the correlation data.

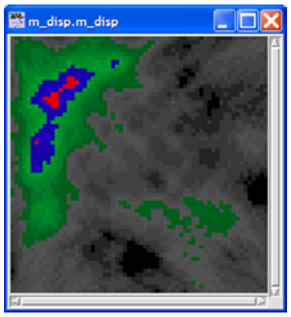
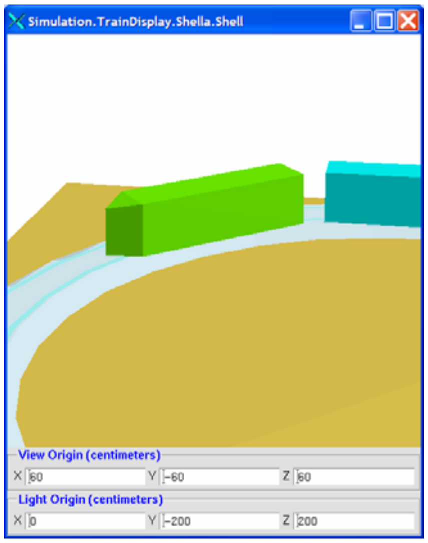


Figure 2 - The correlation of the four audio channels forms a spot of high intensity.

Simulation

The application was first developed as a simulation. The environment of the train and camera was simulated, and the four channels of audio data for the microphone array were read from files. To show the results of the simulation, a 3D rendering of the scene is presented from the view angle of the camera, as shown in Figure 3.



View Origin (centimeters)
X 60 Y -60 Z 60
Light Origin (centimeters)
X 0 Y -200 Z 200

Figure 3 - The simulation includes rendering of a 3-D model of the environment.

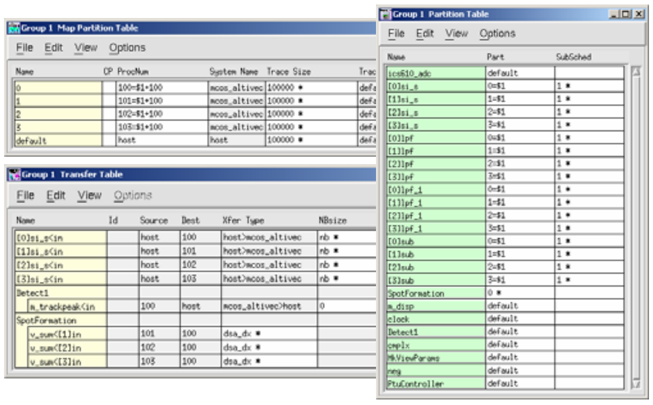
Using Gedae-Simulation, experiments were done on multiprocessor implementations of the application to prepare for moving it to hardware processing realtime data.

Once created, the code of a Gedae application does not have to be changed in order to partition and map it to multiple processors. During simulation, several mappings to virtual processors were used, in different configurations, and the results were analyzed in the Gedae Trace Table.

Multiprocessor DSP Implementation

To transition this application to using real world data, we ported it to the Mercury AdapDev system. The Mercury AdapDev system provides an Intel Pentium host and two quad DSP boards where each DSP is a 500MHz AltiVec processor. Physical components for the camera, gimbal, microphones, and audio digital converter (ADC) were assembled. The application was altered to remove the artificial audio source and scene rendering and replace it with an interface to the ADC (via PCI), the gimbal (via serial port), and the camera (via USB).

While the sources and sinks were replaced to use real world data, the algorithms and their coding did not need to be changed to create a realtime implementation. Using our experiments from the simulation, a partitioning and mapping scheme was entered into the Partition and Map Partition Tables shown in Figure 4. The communication protocols were tweaking using the Transfer Table, picking direct schedule access transfers (equivalent to DMA) and removing the blocking of the host-to-DSP transfers. Additionally, automated stripmining was used to optimize vectorization and improve cache utilization. These changes, include both changing the graph to use real world data and setting the implementation parameters, took one day of effort and the resulting implementation is able to process three frames per second – sufficient to track the train at its maximum speed with a low number of errors or jitter.



Name	CP	ProcNum	System Name	Trace Size	Trac
0		100=$1+100	mcos_altivec	100000 *	defa
1		101=$1+100	mcos_altivec	100000 *	defa
2		102=$1+100	mcos_altivec	100000 *	defa
3		103=$1+100	mcos_altivec	100000 *	defa
default		host	host	100000 *	defa

Name	Id	Source	Dest	XFer Type	NBsize
[0]si_s<in		host	100	host>mcos_altivec	nb *
[1]si_s<in		host	101	host>mcos_altivec	nb *
[2]si_s<in		host	102	host>mcos_altivec	nb *
[3]si_s<in		host	103	host>mcos_altivec	nb *
Detect1					
m_trackpeak<in		100	host	mcos_altivec>host	0
SpotFormation					
v_sum<[1]in		101	100	dsa_dx *	
v_sum<[2]in		102	100	dsa_dx *	
v_sum<[3]in		103	100	dsa_dx *	

Name	Part	SubSched
ics610_adc	default	
[0]si_s	0=$1	1 *
[1]si_s	1=$1	1 *
[2]si_s	2=$1	1 *
[3]si_s	3=$1	1 *
[0]lpf	0=$1	1 *
[1]lpf	1=$1	1 *
[2]lpf	2=$1	1 *
[3]lpf	3=$1	1 *
[0]lpf_1	0=$1	1 *
[1]lpf_1	1=$1	1 *
[2]lpf_1	2=$1	1 *
[3]lpf_1	3=$1	1 *
[0]sub	0=$1	1 *
[1]sub	1=$1	1 *
[2]sub	2=$1	1 *
[3]sub	3=$1	1 *
SpotFormation	0 *	
m_disp	default	
clock	default	
Detect1	default	
cmplx	default	
MkViewParams	default	
nmg	default	
PtuController	default	

Figure 4 - Application was mapped to multiple DSP processors without changing the code.

Multicore Implementation

To illustrate support for the Cell/BE, this application was ported to the Sony Playstation 3. The Cell/BE on the Sony Playstation 3 provides a dual threaded Power Processing Element (PPE) core as well as six Synergistic Processing Elements (SPE), as shown in Figure 5. The SPEs are very efficient vector processors but have very tight memory restrictions, with only 256KB of local storage and no cache. Programming the Cell/BE by hand requires careful management and planning of memory and data movement between the SPEs.

Gedae addresses the issues of memory management and data movement directly. The automated implementation of these issues simplifies development for the Cell/BE. After altering the application to use a USB-based ADC (the Playstation 3 does not have a PCI slot), the application was easily moved to the Cell/BE, and the process of optimizing it for the multicore architecture took two hours.

To optimize the application for the Cell/BE, the compute-intensive signal processing portion of the application is partitioned for the six SPEs. The memory footprint of the program and data is taken into account during this process, using the Schedule Parameters dialog to analyze the size of the threads that will be created for each partition. To reduce the size of the memory footprint, the automated stripmining capability is used, allowing a set of audio vectors to be processed independently on each SPE instead of en masse. Additionally, a primitive that performs a column-wise sum of a matrix is identified as pushing the thread memory size over the limit. To fix the issue, the primitive is replaced with one that integrates a series of row vectors.

The Gedae Trace Table is used to analyze the performance when running on the Cell/BE. During this process, one primitive wass identified as being slow, and it was recoded to use a unity stride. Based on the processor load, the distribution of the work was also altered. After two hours, in the final optimization, four SPEs are used to do a majority of the preprocessing (one SPE per audio channel), including the band filtering of the frequency spectrum. The other two SPEs are used to combine the data in the correlation calculation of the spot formation. The PPE performs the detection algorithm and interfaces with the I/O devices. With this implementation, the application is able to process almost 15 frames a second on the Cell/BE, providing a much smoother tracking of the train.

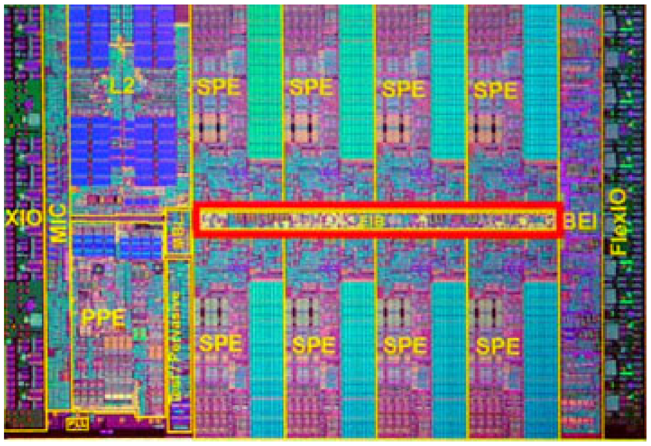


Figure 5 - The Cell/BE provides a PPE core and 8 SPE cores (6 enabled on the Playstation 3).