Mapping an Application to RACE® i860 Compute Environments

Application Note 201.0

TARGET AUDIENCE

Engineers and programmers interested in the design considerations associated with developing RACE multicomputer applications.

ABSTRACT

This note examines how to map a single application to Mercury's RACE multicomputer architecture using i860-based Compute Environments (CEs). It discusses how to determine the number of CEs necessary to process an input data stream in realtime, how to divide the input data stream, how much memory is needed, how much memory bandwidth is consumed by I/O, and the inter-process synchronization requirements. The application used to illustrate these concepts is the convolution of a fixed kernel with a continuous data stream.

Authored by:
Brian Bouzas, Systems Engineering

REVISION HISTORY

Revision #	Description
201.0	Creation

Mapping an Application to RACE i860 Compute Environments

This note examines how to map a simple application to Mercury's RACE multicomputer architecture using i860-based Compute Environments (CEs). It discusses how to determine the number of CEs necessary to process an input data stream in real time, how to divide the input data stream, how much memory is needed, how much memory bandwidth is consumed by I/O, and the inter-process synchronization requirements.

The application used to illustrate these concepts is the real convolution of a fixed kernel with a continuous data stream of unspecified length. The kernel contains 400 elements and the input data stream consists of 16-bit signed integers input at a rate of 50 Million Samples per Second (MS/s) via Mercury's RIN-T direct input daughtercard. The design goal is to process the input stream in real time with the minimum number of i860 CEs (see Figures 1 and 2).

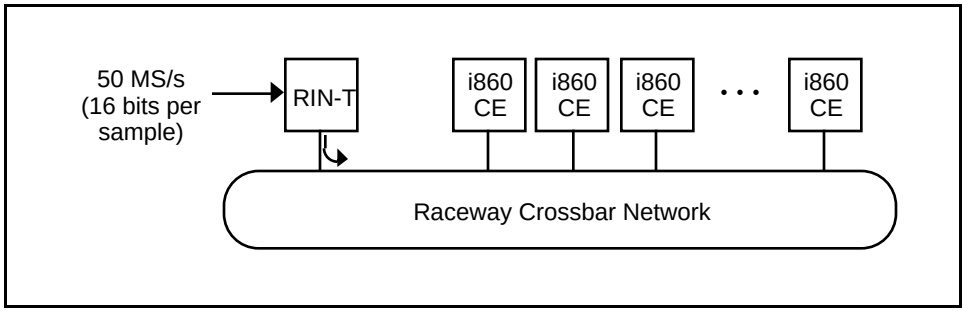


Figure 1. RACE system with a single RIN-T input device and multiple i860 CEs.

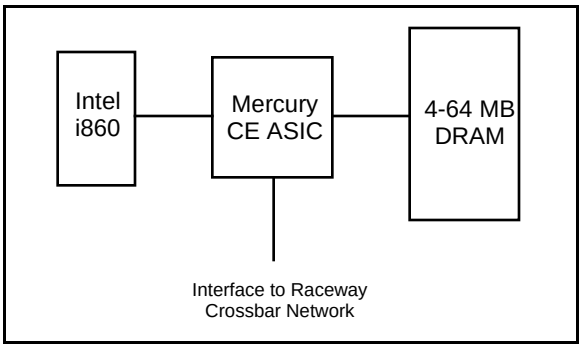


Figure 2. RACE i860 Compute Environment. Mercury's i860 CE is clocked at 40 MHz and has a DRAM memory bandwidth of 160 MB/s.

Mapping the algorithm to multiple processors

To perform time-domain convolution, a RACE i860 CE requires approximately 1.1 clock cycles per filter tap for each output data point (see timings published in the RACE i860 Scientific Algorithm Libraries data sheet). A 400-tap filter, therefore, requires 440 **C**lock cycles **P**er output **P**oint (cpp), allowing a single i860 CE to process at most 91,000 samples per second (40 MHz/440 cpp). If the input data stream were divided among multiple CEs and portions of the data set were processed in parallel, over 550 i860 CEs would be needed to keep up with the 50 MS/s input rate (50 MS/s ÷ 91,000 samples/sec per CE).

Because direct time-domain convolution is so computationally intensive, continuous convolution with a kernel of 400 elements is best performed in the frequency domain by taking advantage of the convolution property of Fourier transforms and the computational efficiency of FFTs. Specifically, the circular (or periodic) convolution of two finite sequences can be obtained by multiplying the discrete Fourier transforms of the two sequences and taking the inverse discrete Fourier transform of the result. Using the technique known as sectioned convolution, this property of Fourier transforms can be applied to the case where one sequence is essentially infinite in length.

With sectioned convolution, the input stream is divided into N-point sections where each section overlaps the next by the size of the kernel, K. Circular convolution is performed on each section individually by multiplying the discrete Fourier transform of the input signal with the discrete Fourier transform of the convolution kernel and taking the inverse discrete Fourier transform of the result. The first K elements from each of the individual section convolutions are discarded, and the remaining output elements from each section are assembled to form the desired linear convolution[1] (see Figure 3).

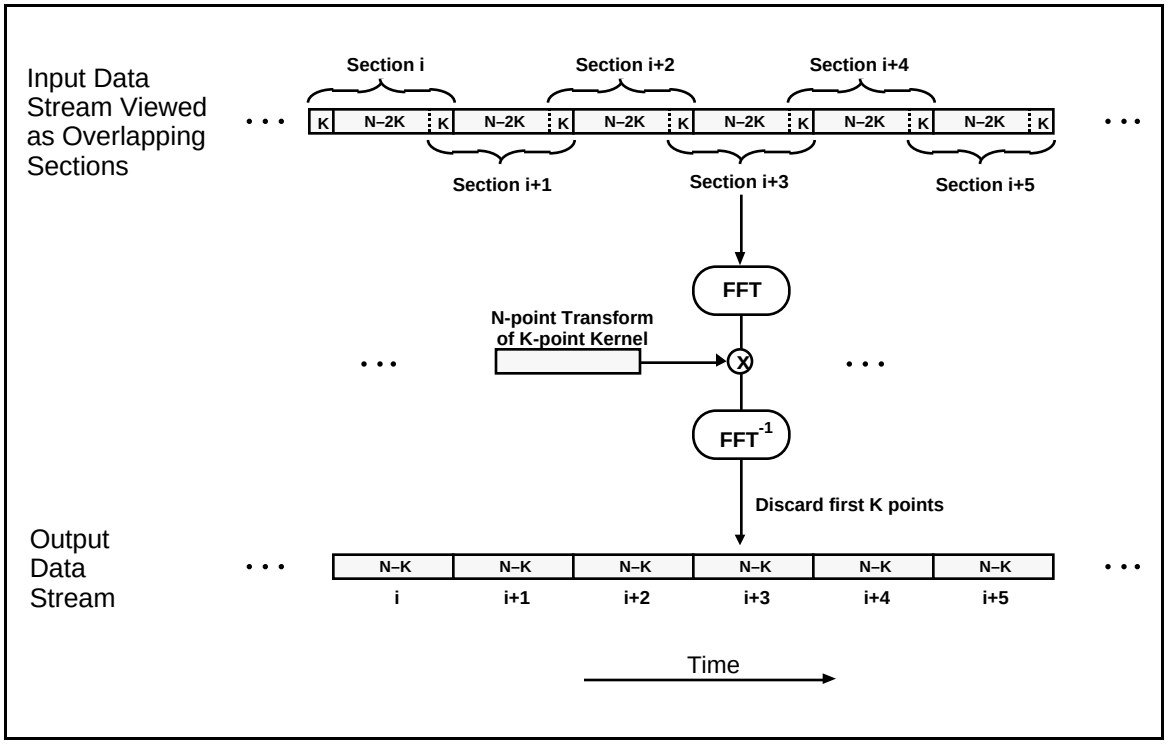


Figure 3. Sectioned Convolution. Each N-point section of the input stream is convolved with the same K-point kernel using multiplication in the frequency domain. The convolution from each section yields N–K points in the output data stream. (For the example discussed here, K=400.)

Table 1 summarizes the processing steps required for each piece of the sectioned convolution. The integer to floating point conversion (Step 1) is necessary in order to use the i860 processor efficiently. Although it can perform integer arithmetic, the i860 is optimized for processing vectors of floating point numbers. The Fourier transform of the fixed kernel used in Step 3 is not treated as a separate processing step because it is computed only once. Before the transform of the kernel is computed, the K elements are zero padded out to the section length. Note also that because the input data is real, only N/2 complex multiplies are required to accomplish Step 3.

[1]Discarding the first K elements eliminates the aliased values generated by the circular convolution.

Step	Action	Scientific Algorithm Library Function
1.	Convert input integers to floating point format.	vflt()
2.	Perform forward real FFT.	rfft()
3.	Multiply output of Step 2 by FFT of convolution kernel.	cvmul()
4.	Perform inverse real FFT.	rfft()

Table 1. Processing sequence for circular convolution with the RACE i860 CE.

Although sectioned convolution is less computationally demanding than direct time-domain convolution, a single i860 CE cannot process the entire 50 MS/s data stream in real time. In order to keep up with the input data rate, the sectioned convolution algorithm can be distributed across multiple CEs by having the RIN-T distribute the input data stream among CEs in a round-robin fashion, giving some number of sections to each CE (see Figure 4).

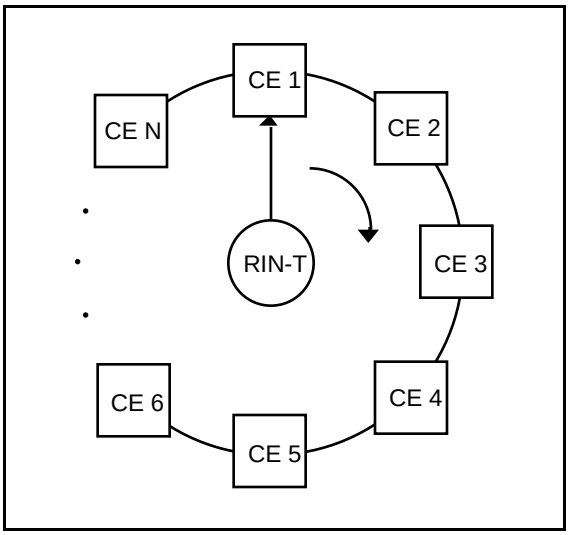


Figure 4. Round-robin data distribution from a RIN-T to N CEs.

Because the RIN-T is a stream input device, however, it can deliver each element of the input data set to a single destination only. Therefore, some amount of communication is required between CEs to accommodate the overlap region between sections. Assuming the RIN-T is programmed to change destination CEs at the end of a section, each CE must share the last K elements it receives from the RIN-T with the next CE in the round-robin chain (see Figure 5).

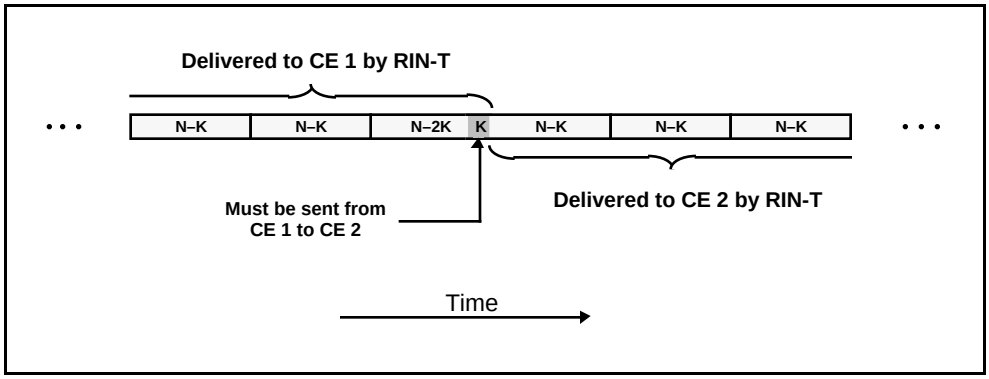


Figure 5. Data distribution boundary. A K-point overlap region must be shared between CEs.

Sizing the target system

Once a general approach for mapping the application has been established, there are four basic steps to determining the number of CEs required. These steps are listed in Table 2. The remainder of this user note applies these steps to sectioned convolution. Although the focus of this user note is to maximize real-time throughput, the steps in Table 2 can be applied regardless of the design criteria (e.g., if latency rather than throughput is the primary concern).

Steps for Determining the Number of CEs Required
1. Evaluate the interaction between relevant parameters and choose a set of parameters that makes efficient use of processing power, I/O bandwidth, and memory.
2. Determine the processing load per CE without I/O.
3. Determine the impact of I/O (both external and between CEs) on a single CE.
4. Calculate the total number of CEs required.

Table 2. Steps for determining the number of CEs.

How much data should be given to each CE?

By putting the largest possible buffers on each CE, the data movement and synchronization associated with sharing the overlap data will be minimized. Assuming the smallest standard memory size of 4 MB per CE and allowing 1 MB for application code, static data structures, and the MC/OS operating system (a conservative estimate), 3 MB are available for data buffers. Allowing 2 bytes per point for an input buffer and 4 bytes per point for a buffer of floating point data, each CE can easily work on 512K input points.

Synchronization between CEs is accomplished quickly (i.e., in less than a couple of microseconds) by using flags in DRAM to indicate that the overlap data has been delivered. This synchronization technique takes advantage of each CE's ability to access the memory associated with another CE directly, without intervention from the remote CE's processor.

What is the best section size?

The section length must be a power of two in order to compute the discrete Fourier transform with a standard FFT routine. The minimum section size is the nearest power of two greater than twice the kernel size; the maximum section length depends only on the amount of memory available. With a kernel size of 400 elements and an input buffer size of 512K points, the section length (and hence the FFT size) can vary from a minimum of 1K points to a maximum length of 512K points. The larger the section length, the smaller the percentage of each FFT that must be discarded due to overlap.

The optimum section length can be determined by comparing the effective number of clocks per point necessary to perform the convolution when the 400-point overlap between sections is taken into account. Table 3 shows that for the i860, 2K-point sections with 19.5% overlap are most efficient. Although larger sections have the advantage of discarding a smaller percentage of output points, 2K-point real FFTs make the best use of the i860's 2K-point (8 KB) data cache.

Section Length	Clocks per Point with No Overlap	% Overlap with 400-point kernel	Effective Clocks per Point with Overlap
1K	35.4	39.1%	49.2
2K	38.8	19.5%	46.4
4K	55.6	9.8%	61.0
8K	57.2	4.9%	60.0
16K	61.0	2.4%	62.5
32K	68.8	1.2%	69.6
64K	78.2	0.6%	78.7

Table 3. Convolution processing requirements in clocks per point for various section lengths with 400-point overlap. Processing consists of Steps 2-4 from Table 1; Step 1 is not included since it is not affected by overlap.

What is the processing load for a single CE ?

Table 4 gives the number of clock cycles per output point required for each stage of the processing on each CE. Using a section length of 2K points and neglecting the effects of I/O, processing takes 50 clock cycles per output point. Each i860 CE is, therefore, able to process 0.8 million input points per second (40 MHz/50 cpp) requiring a minimum of 63 CEs to process the input data stream in real time (50 MS/s ÷ 0.8 MS/s per CE).

	Clocks per Point
Short to Float Conversion	3.6
Forward Real FFTs	20.6
Complex Vector Multiply	4.4
Inverse Real FFTs	20.6
Total	50

Table 4. Processing breakdown for a single i860 CE using a section length of 2K points.

What is the impact of I/O?

Once the processing load is known, the impact of I/O on processing should be taken into account. Each Mercury CE is designed to allow efficient processing while I/O occurs simultaneously in the background. Because data is moved into CE memory directly by the RIN-T, or between CE memories using the DMA controller at each CE, no CPU cycles are required for data movement. However, since the memory bandwidth used by the external I/O is unavailable for local processing, processing efficiency may be decreased by the percentage of memory bandwidth consumed by those external accesses (Refer to the Application Note entitled *Concurrent I/O and Application Processing in the RACE Architecture*, for a complete discussion).

A 100 MB/s input stream (i.e., 50 MS/s) divided across 63 CEs results in 1.6 MB/s of input to each CE. The I/O associated with sharing the overlap region is negligible, on the order of 1 kB/s (800 bytes/1 MB x 1.6 MB/s). A 1.6 MB/s I/O rate has only a 1.0% impact on processing efficiency, allowing each CE to process with 99% efficiency:

$$\%\ \text{Processing Efficiency} = \left(1 - \frac{\text{I/O Rate in MB/s}}{160\ \text{MB/s}}\right) \times 100$$

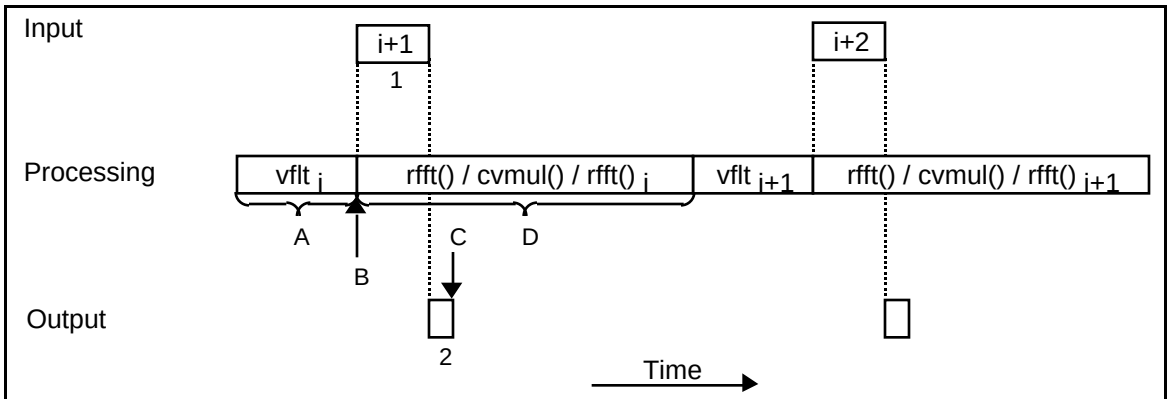
How many CEs are required?

With a processing efficiency of 99%, a 50 cpp task requires 51 cpp (50 cpp / 0.99), and the minimum number of i860 CEs becomes 64:[2]

$$\frac{50\ \text{MS/s} \times 51\ \text{cpp}}{40\ \text{MHz/CE}} = 64\ \text{CEs}$$

Processing Timeline

To maximize throughput, the input data should be double buffered so that the i860 processes a buffer while the RIN-T supplies data for the next buffer. Figure 6 shows a processing timeline and indicates how I/O is overlapped with processing. Note that the arrangement shown does not require any additional memory for "double buffering" since the input buffer can be re-filled once it has been converted to floating point format. Although not critical for this application which has a relatively low I/O rate per CE, it is best to overlap I/O with processing that makes heavy use of the i860's data cache (e.g., FFTs). By using the DRAM memory when the processor does not need it anyway, external accesses have even less impact on local processing.



Processing Notes

A. Convert all 512K points from input buffer i to floating point format. (Step 1 from Table 1)
B. Notify RIN-T that the input buffer is free.
C. Notify CE N+1 that the overlap region has been delivered.
D. Process buffer i. (Steps 2-4 from Table 1)

I/O Notes

1. Receive buffer i+1 from the RIN-T (plus overlap from CE N-1).
2. Output overlap points for buffer i+1 to CE N+1.

Figure 6. Processing timeline for a single CE.

Summary

Using a simple application this note has illustrated many of the design issues that must be considered when mapping an application to RACE i860 Compute Environments. These issues include splitting the computation across multiple processors, individual processor loading, DRAM memory bandwidth, external I/O, and inter-processor synchronization.

[2]This minimum number of CEs applies only to the steady-state case and leaves no margin for additional processing. When building a system, an appropriate "safety margin" should be factored into the design.