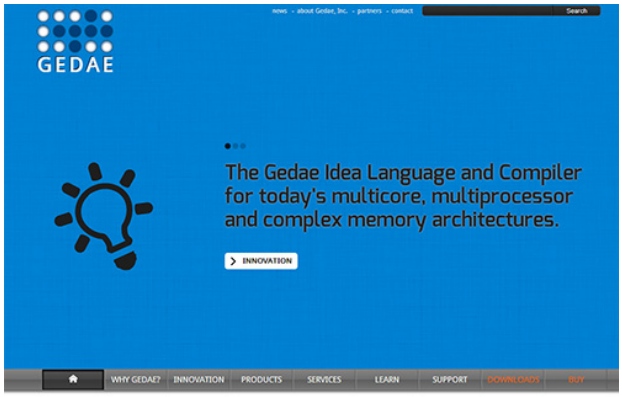
Issue No. 5, August 2013

G E D A I C A

pl n 1.of or relating to the culture, artifacts, customs of Gedae



The Gedae Development Environment

Gedae is a software development environment (see figure) with a language and compiler that automatically builds software for today's complex computer systems. Users benefit from the staggering effects automation. Productivity is increased tenfold because the tedious details are implemented by the

AND WE'RE BACK...

After a Brief Hiatus to Create a New Website!

If you haven't checked it out yet you should. Everyone else is doing it! Seriously we've been overwhelmed and grateful for the response. With over 1000 unique vistors in the first two weeks it seems like a hit.

Check out a few interesting pages:

- The Gedae Technology
- The Gedae Product Page
- Gedae's Impact
- News
- About Gedae, Inc.

Please send your suggestions, questions or comments to jlundgren@gedae.com.

GEDAE IS GROWING...UP

We're Updating Our User Interface!

We're still in the nascent stages of the update but wanted to share our excitement with you!

As with the website, our goal is to simplify & clarify in order to amplify the value of Gedae. Our goal has always been and continues to be to give developers and companies the best available tools for the development of powerful applications for today and tomorrows complex computer architectures. We aim to do that by creating technology and tools that are transparent and enhance your abilities - not get in the way.

The Road Map

First Release - EXPECTED RELEASE DATE: 4Q 2013

The first release will offer a streamlined and simplified version of the existing UI. It will also included updated, modern look and feel and improved support and help features.

Second release and beyond plans include: migrating to native windowing systems, increased integration with ecosystem tools, alternate Workspace set-ups for different user roles and tasks and much, much more...

Be sure to check the ROADMAP page on the website to learn more as it becomes available!

CALLING ALL USERS!

We're seeking long time Gedae users to participate in shaping the new Gedae Development Environment. Contact us via e-mail at werelistening@gedae.com to participate.

ON THE ROAD AGAIN

Interested in learning more about how Gedae can revolutionize your orgainzation?

Participate in our upcoming roadshow. We'll be on the road demonstrating the results of our joint Intel/Kontron/NASoftware SARMTI Benchmark (see more on page 2), presenting the latest Gedae products and technology and our roadmap.

Contact us via email at roadshow@gedae.com to learn more and set up a visit.

DOLLARS AND SENSE

A long time Gedae customer benefited from the code portability enabled by the automation in Gedae's compiler by writing their code once and reusing it to bid on multiple programs. The code reuse reduced their cost to create a bid by 50%. Additionally, they confidently bid at lower cost knowing they could still meet their profitability goals. These benefits enabled them to cast a wider net, capture programs (and revenue) and increase their profit.

That's how one Gedae customer benefited from code reuse. What can you do with increased leverage over your intellectual IP?

Learn more about increasing your leverage over your intellectual IP from our upcoming white paper: "Using Gedae as an Algorithm Repository". Check our website for details.

CODE, TOOLS & TIPS

SARMTI DEMONSTRATION [website link]

Is there an example you'd like to see in CODE, TOOLS & TIPS? — We're looking for industry parallel processing applications to implement in Idea Language and publish results. Submit yours today via cc@gedae.com.

The example described here illustrates the ease of implementing efficiency strategies in the Gedae Development Environment. It also implements a general principle that needs be applied to every effort to build efficient software running on parallel processing cores. The SARMTI algorithm developed by NA Software uses a data cube. The cube is shown in figure [1] and described by the expression:

[1]
```
x[b][r][c]
```

where b,r and c are range variables and used here as array indices. The third index we will refer to as a plan of data. The first operation is to collapse the data in this case by summing all the planes of data. The

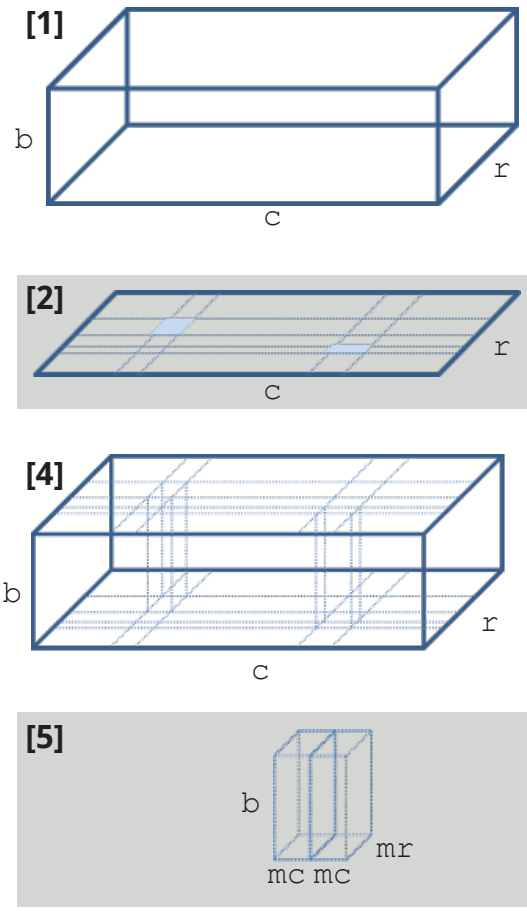
[2]
```
x1[r][c] += x[b][r][c];
mask[r][c] = x1[r][c]>K;
```

where += is a collapsing operator and K is a parameter. Since it uses a plus then it sums over the range variable missing from the left hand side of the assignment statement, in this case `b`.

[3]
```
clrng[r][c], r0[n], c0[n], r1[n], c1[n] = bwlabel(x1);
sz_r[n] = r1[n] - r0[n] + 1;
sz_c[n] = c1[n] - c0[n] + 1;
mr >?= sz_r[n]; // max collapsing operator
mc >?= sz_c[n];
```

where bwlabel colors the regions created in the mask from 1 to #n where #n is the size of the range variable n and returns the rectangle that delineates the region of interest (ROI). Step 4 shows the regions of interest (ROIs) in the data cube. In step [5] the ROIs are collected and processed.

[5]
```
roi[n][b][mr][mc] = x[b][r0[n]][c0[n]][mr][mc];
for(n1=0, Target target[n] = 0; n1<#n; n1 = n1 + 1) {
  range rr = sz_r[n1];
  range cr = sz_c[n1];
  roi1[b][rr][cr] = roi[n1][r0[n1]+rr][c0[n1]+cr];
  roi2[b][rr][cr] = ProcessPlanes(roi1);
  targ = MkTarget(roi2,r0[n1],c0[n1]);
  target[n] = set(targ,n1);
}
```



where Target is an Idea language data structure customer created for this application. In the first Idea implementation the function ProcessPlanes() was run on the full data cube. This implementation, a sparse image implementation, is much more efficient. Further, ProcessPlanes() requires a cornerturn of the data cube. In the parallel implementation the corner turn requires a shuffling of data among the processors and puts a high load on the data bus. The version shown here does the transpose locally. The timing results of the version hand-coded by experts and the version coded in Gedae is shown in the table below. Using the technique described above reduced the load on the data bus and allowed the near linear speed up shown. The hand-coded version used a different parallelization scheme than the Gedae version. A primary reason for being able to implement the more complex version in Gedae is that 9000 LOCs of hand written code was reduced to 750 LOC in the Idea language. This change in LOC is a direct result of the automatic implementation of the software by the compiler. Note than even on one quad core processor the Gedae version was faster.

Hand Coded Timing Results [in seconds]		Demonstration #				
		1	2	3	4	5
[N] Procs x [M]Cores	1x1	6.8	14.2	28.3	72.6	285.5
	1x4	2.05	4.42	8.61	21.5	87.6
	2x4	1.17	2.39	4.73	12.0	48.6

Gedae Timing Results [in seconds]		Demonstration #				
		1	2	3	4	5
[N] Procs x [M]Cores	1x1	4.00	8.92	17.4	44.3	172.4
	1x4	1.24	2.79	5.52	13.9	57.8
	2x4	0.71	1.54	3.05	7.67	31.8