Efficient Processing of Multiple Contacts in MPP-DYNA

Brian Wainscott
Livermore Software Technology Corporation
7374 Las Positas Road, Livermore, CA, 94551 USA

Abstract

Complex models often contain more than just a few contact interfaces. The decomposition of such a model can result in an uneven distribution of these contacts among the available processors. Some contacts may lie wholly on a single processor, while others will be distributed across many or all of the processors.

Some processors may have many contacts to handle, and others may have none. This variability can cause inefficiencies which adversely impact scalability. I will show recent work on contact algorithms in MPP-DYNA which addresses some of these issues.

Introduction

One of the primary measures of efficiency for an MPP program is scalability. Increasing the number of processors used on a problem should substantially decrease the runtime – ideally in proportion to the number of processors used. The primary reasons why this ideal might not be achieved as the number of processors increases are increasing communication costs (time the processor spends doing communication instead of calculation) and load imbalance (time the processor spends waiting for another processor). Load imbalance can be further divided into computational imbalance and processor idling. Computational imbalance here is taken to mean that some processors have more calculation to do than others, and so take longer. The nature of time step integration as used in MPP-DYNA requires a certain amount of synchronization between the processors, and unequal computational load results in some processors waiting. But there are other kinds of synchronization issues that can result in processor idling, and one of these is contact.

Problem

The standard implementation of contact in MPP-DYNA can be thought of as treating one contact interface at a time. Certainly this is true at the processor level: each processor will handle the contacts it is involved in, in some predetermined order. Each contact interface may be spread across several

processors, and these processors will coordinate the proper parallel implementation of that contact interface. But if there are multiple contact interfaces in the model, processor idling can occur even if every processor has the same amount of work to do.

For example, consider a problem that has 3 contact interfaces and is run on 3 processors. If, for whatever reason, the contact interfaces are distributed across processors as shown in table 1, each processor will be idle during the processing of one of the contact interfaces, as it waits for the other two processors.

	Processor 0	Processor 1	Processor 2
Contact 1	X	X	-
Contact 2	-	X	X
Contact 3	X	-	X

Table 1: X represents processors involved in a contact interface, - indicates a processor is not involved

There is no forced synchronization here on a per contact basis, some waiting is just unavoidable. For example, upon entering the phase of the time step where contact calculation is performed, processor 2 immediately begins work on contact 2 (it does not "wait" for contact 1 to finish, and in fact does not necessarily know there is a contact 1). But one of the first things processor 2 must do is communicate with processor 1, which is busy working on contact 1. With this approach, this kind of idling is unavoidable. When the number of contact interfaces and number of processors get large, this order dependent idling contributes to poor scalability.

Solution

I developed new contact algorithms for MPP-DYNA to address this issue, the so-called "groupable" contact routines. ("Groupable" because they can be grouped together for processing, and because the preferred term "parallel" already properly applies to the existing MPP contacts.) In order to give some idea of how the groupable contacts differ from the traditional contact algorithms in MPP-DYNA, we will take a brief look at how these contacts actually work.

In the simplest case, contact comes down to the problem of comparing a single node N with a single contact segment S and applying forces as necessary to insure N does not pass through S. Of course, because of the domain decomposition of MPP-DYNA, there is no reason to expect N and S necessarily exist on the same processor. Supposing N lives on processor 0 and S lives on processor 1, contact will proceed like this:

- Processor 0 sends information about N to processor 1
- Processor 1 performs contact calculations
- Processor 1 sends results back to processor 0

Of course, each processor in reality serves as both a sender and receiver of many nodes at one time, and

things are not quite as straightforward as described here, but this simple view of contact is sufficient for the sake of this discussion. The key difference in the groupable contact is to change the loop ordering from this:

```
Loop over contacts
    Send/receive node data for this contact
    Perform contact calculations
    Send/receive results
End Loop
```

to this:

```
Loop over contacts
    Pack node data for this contact
End Loop

Send/receive node data for all contacts

Loop over contacts
    Perform contact calculations for this contact
End Loop

Send/receive results
```

This has two primary benefits. First, communication overhead is reduced. Rather than sending a separate message for each contact shared by two processors, all the relevant data is packed and sent in a single message. Second, the contact calculations are performed entirely in parallel, without interruption due to communication. There may be some processors having more work to do here than others, so some idling will occur, but the order dependency is broken so this idling is minimized. The result is a significant improvement in scalability.

Examples

We will consider three example problems.

Problem 1: Hemi (Illustration 1)

This is a simple hemispherical deep drawing problem. It has three SURFACE_TO_SURFACE contact interfaces. It is a rather small problem (1599 nodes and 1452 shell elements) that would not be expected to scale very well simply due to its small size.

Problem 2: Manycon (Illustrations 2 and 3)

This is a contrived example built for the purpose of demonstrating the groupable contact. It has 1151856 nodes and 1116160 elements with 496 parts and 75 SURFACE_TO_SURFACE contact

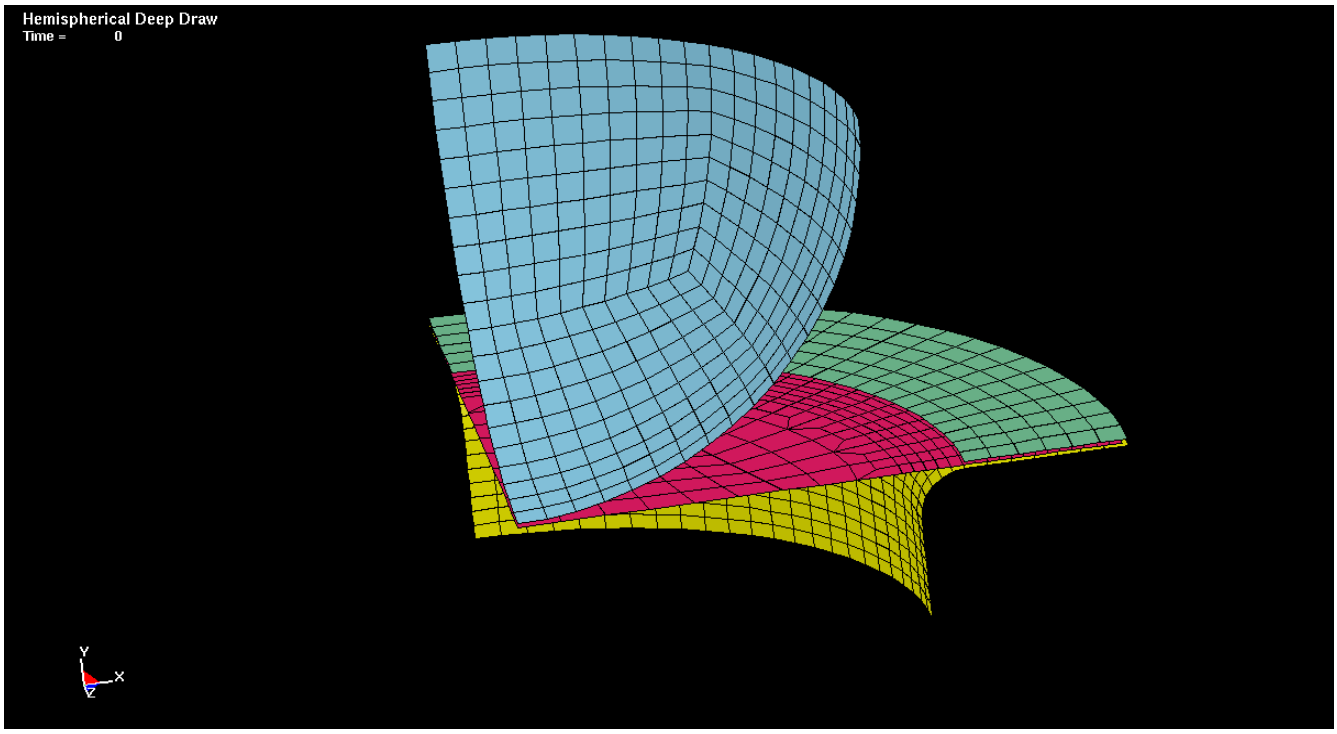

Illustration 1: Hemi problem

interfaces. It contains 5 large plates on the bottom of the stack, on top of which is a 7 x 7 grid of 10 plate stacks, with a final large plate on top (top plate not shown). The bottom plate is fully constrained, and the other plates are constrained in the x and y directions. A pressure is applied to the top plate to push everything into contact. The contact definitions are deliberately distributed in a random fashion around the model: every pair of plates that come in contact occur on opposite sides of one of the 75 contact interfaces, and each interface contains a random collection of about 6 plates on its slave side.

Problem 3: Manufacturing

This is a manufacturing process model from an industrial customer. It has about 43K nodes and about 43K shell elements, and 8 SURFACE_TO_SURFACE contact interfaces.

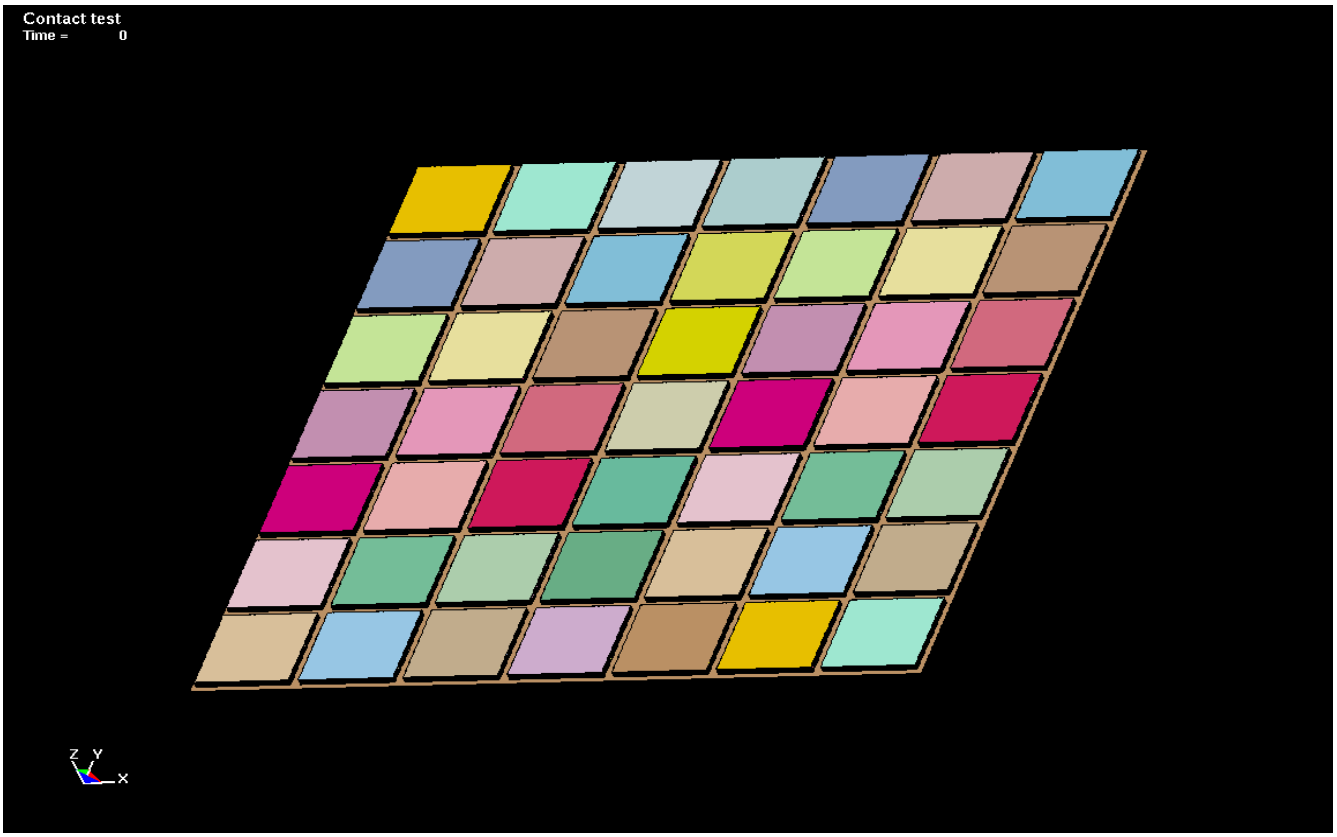

Illustration 2: Manycon problem with top plate removed

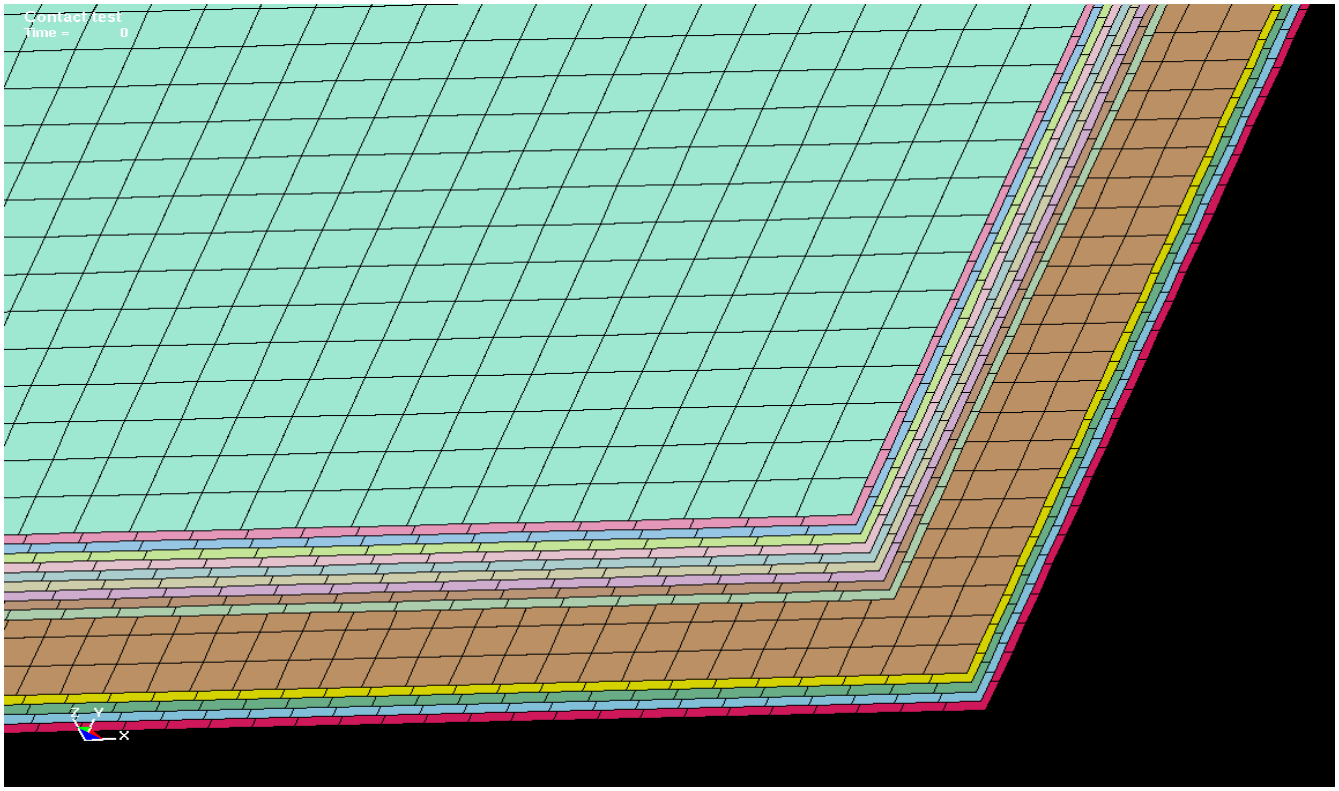

Illustration 3: Manycon problem (close up of corner)

Analysis

To visualize the behavior of MPP-DYNA when comparing the contact algorithms, I have made use of the open source package MPE, which is a time line based tool for analyzing MPI programs. Illustration 4 shows a simple example of output from the Hemi model run on 8 processors using the traditional contact algorithm.

On each end of each time line is a blue circle indicating the start of a time step. The yellow box on the left represents the time spent processing shell elements. The yellow box on the right is the rigid body handling routine. The majority of the time line is taken up by a blue rectangle (contact) with three sub regions,(red, yellow, green) each representing a single contact in the model.

The time spanned by illustration 4 is 5ms. Compare to illustration 5, which also represents 5ms of the groupable contact, and spans nearly 2 and a half integration time steps. In this image there is only a single green region inside the blue contact box. This green region represents the combined processing of the groupable contacts. Most of the variation in the rigid body times represents load imbalance in the contact routines, as there is a synchronization point in the rigid body routines, and processors that arrive there early end up waiting.

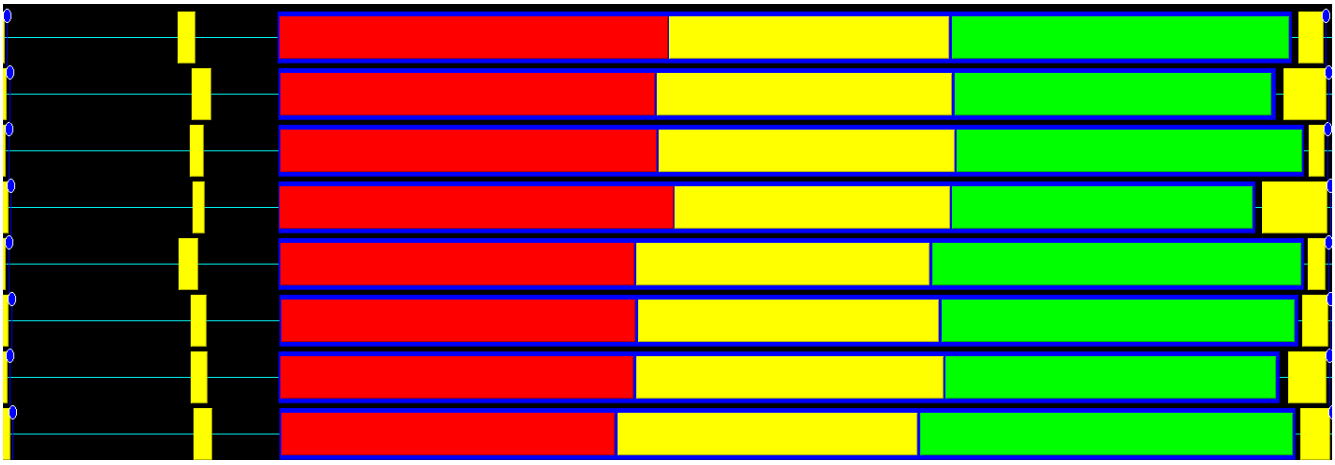

Illustration 4: Hemi 8 processor traditional contact 5ms

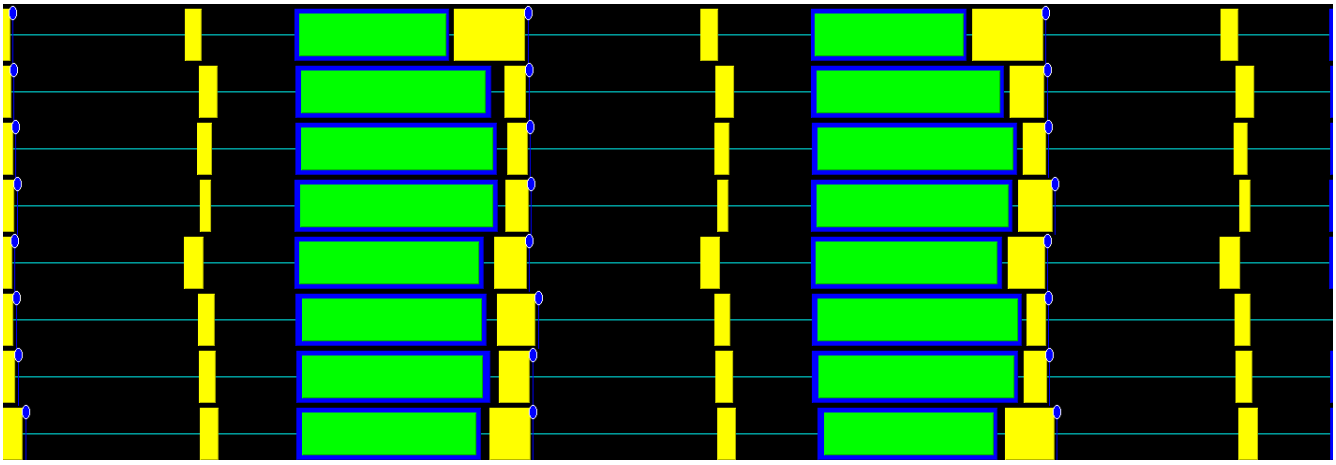

Illustration 5: Hemi 8 processor groupable contact 5ms

Zooming in on these a bit more and turning on the display of MPI messages, we can see how the groupable contact sends fewer messages and so makes more efficient use of the processing power available. (see illustrations 6 and 7, which are not to the same time scale).

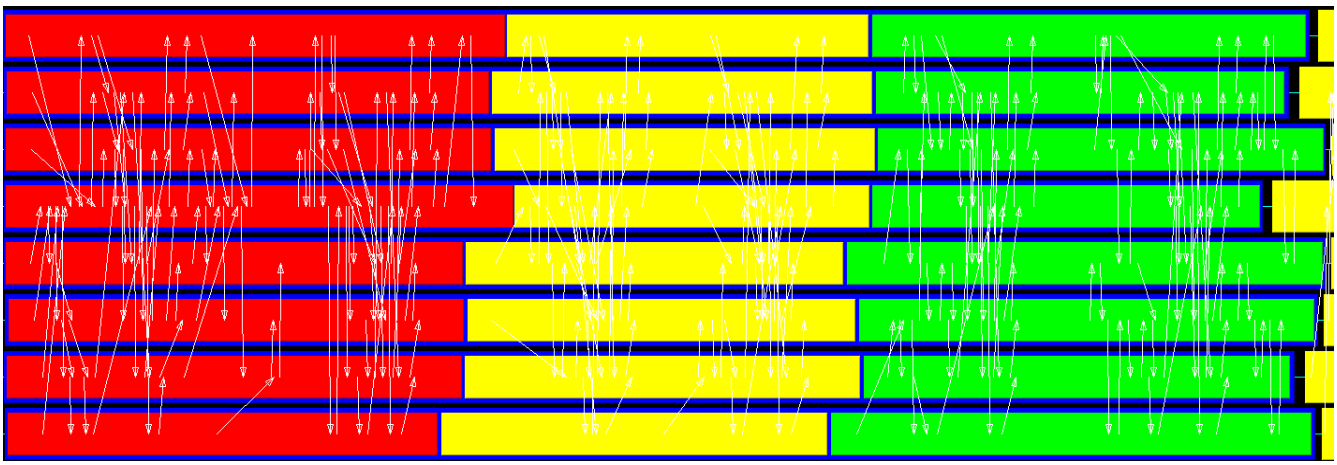

Illustration 6: Hemi 8 processor traditional contact with messages

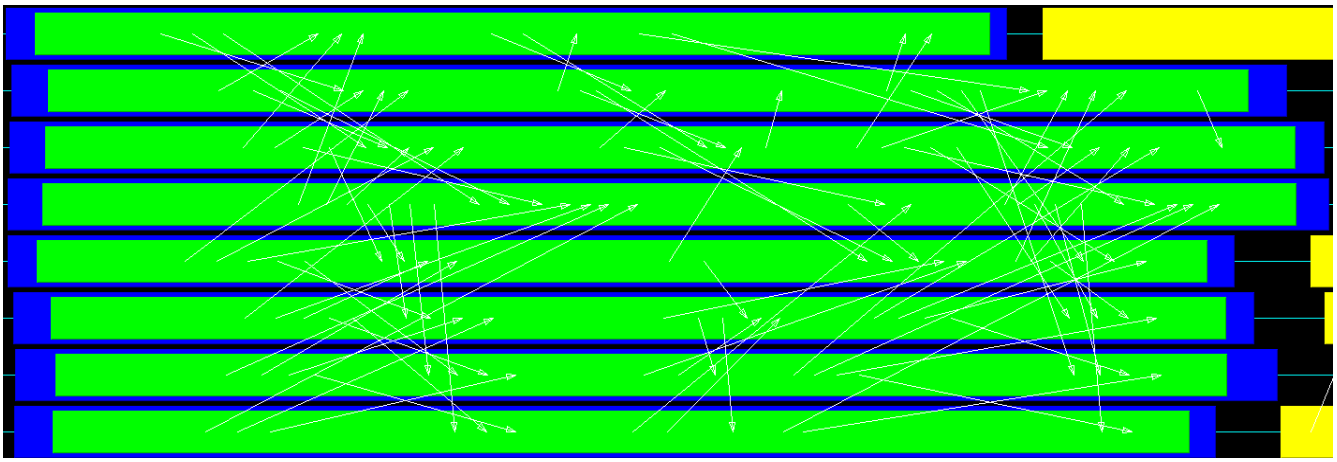

Illustration 7: Hemi 8 processor groupable contact with messages

Similar displays for the Manycon problem are shown in illustrations 8 and 9. These span 150ms. The large number of contacts can be clearly seen in illustration 8, and again the groupable contact is considerably faster. Focusing on just the contact and turning on the display of the messages sent clearly shows the advantage of the new algorithm, where much more of the time is spent doing actual computations (illustrations 10 and 11, which are not to the same time scale).

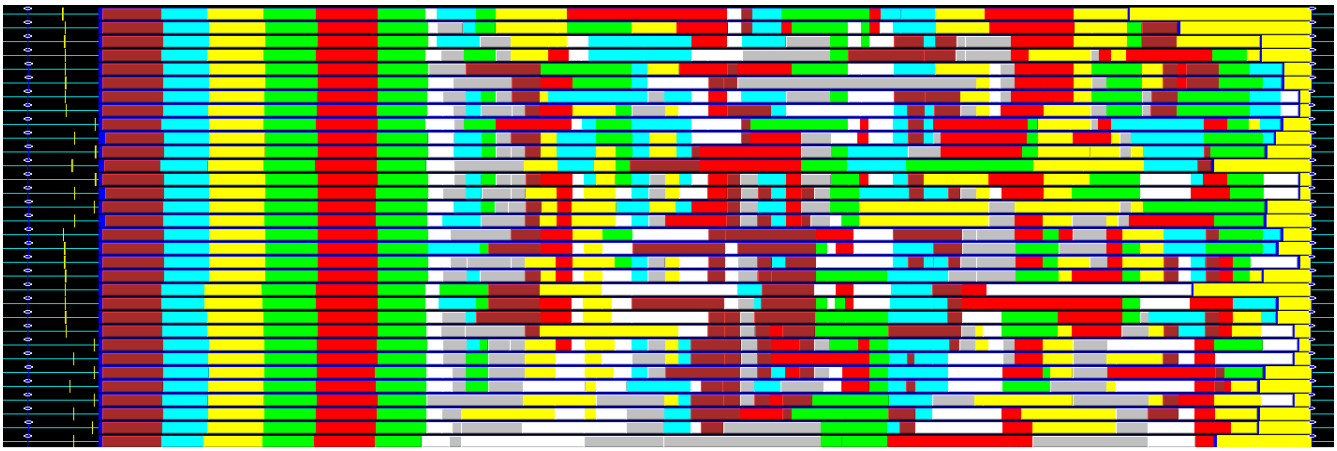

Illustration 8: Manycon 32 processor traditional contact 150ms

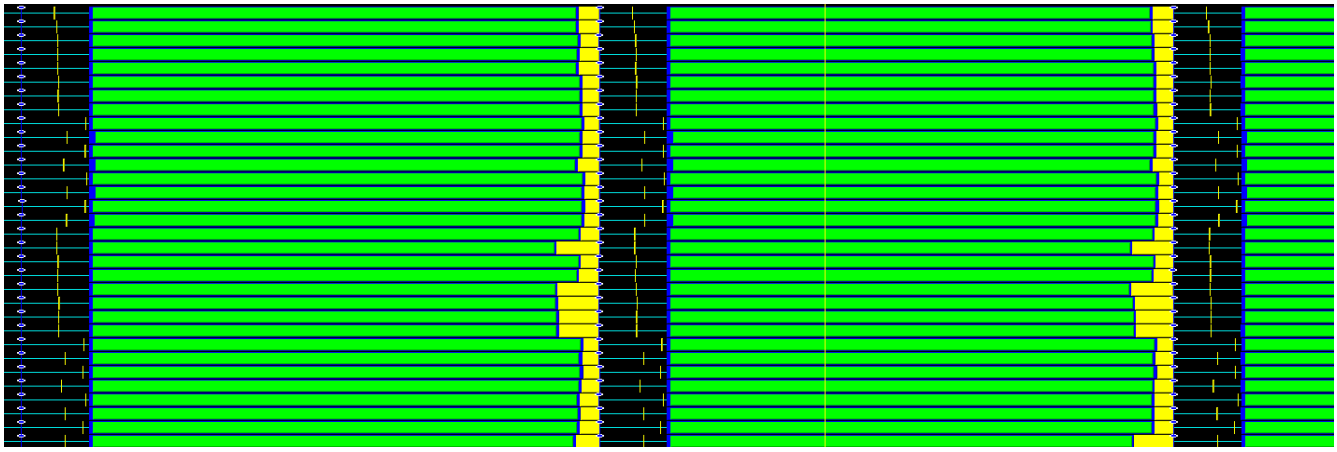

Illustration 9: Manycon 32 processor groupable contact 150ms

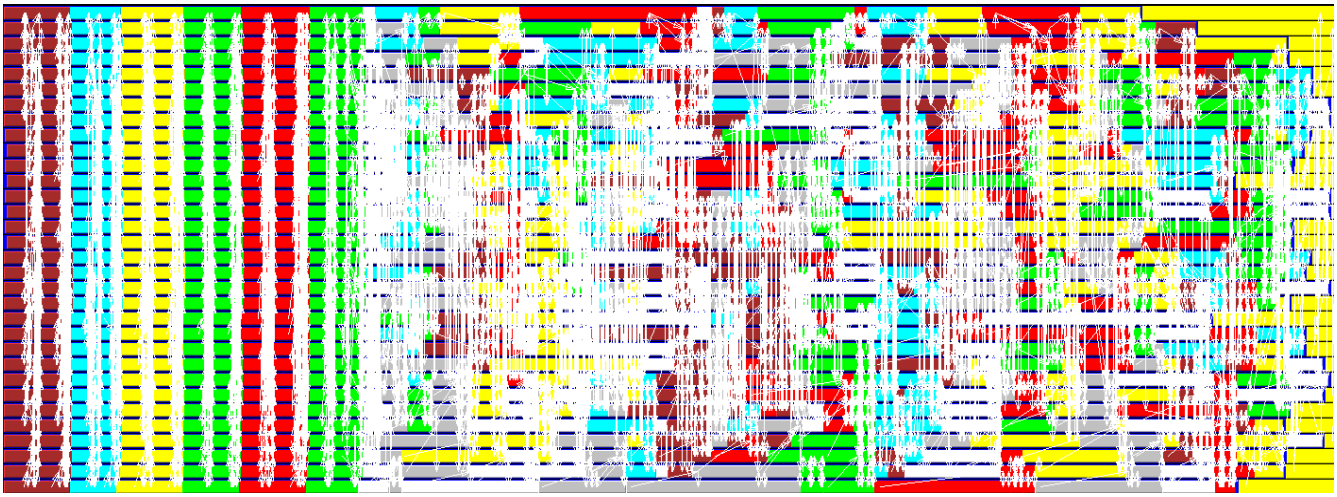

Illustration 10: Manycon 32 processor traditional contact with messages

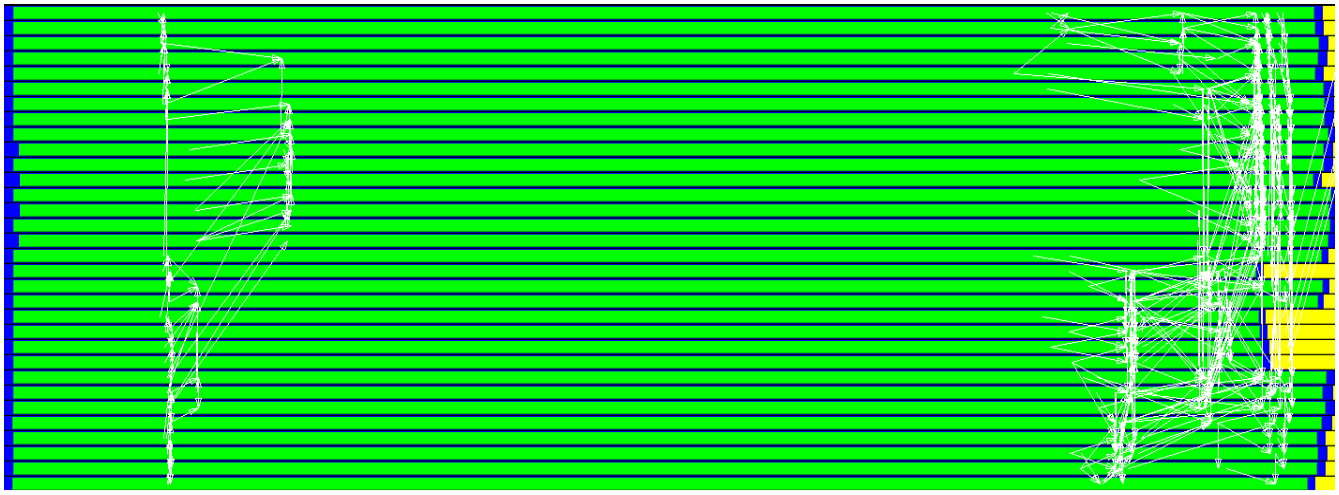

Illustration 11: Manycon 32 processor groupable contact with messages

Results

All of the analyses were done on a cluster at LSTC in Livermore, having 32 compute nodes with the following per node:

- 2 quadcore Intel E5520 Xeon processors @ 2.27GHz
- 48GB of memory
- Local hard disk
- Infiniband interconnect

All problems were run utilizing all 8 cores on each compute node used.

Problem 1: Hemi

Because this problem is so small, it is not reasonable to expect any kind of scalability or speedup on more than a few processors. However, the difference in performance between the traditional and groupable contact algorithms is interesting. The total elapsed runtime of this problem on various numbers of processors is show in illustration 12. On a single processor, the groupable contact is somewhat faster, most likely due to a reduced amount of data copying in the new routines. But what is more interesting is that the new contacts allow for elapsed time improvements all the way out to 16 processors. This is remarkable, as at this point there are only 100 elements per processor, only 33 of which are not rigid.

Problem 2: Manycon

There is very little going on with this problem except contact, which makes it a good test case, and as you can see in illustration 13, the performance of the new algorithm is remarkable. On 16 processors,

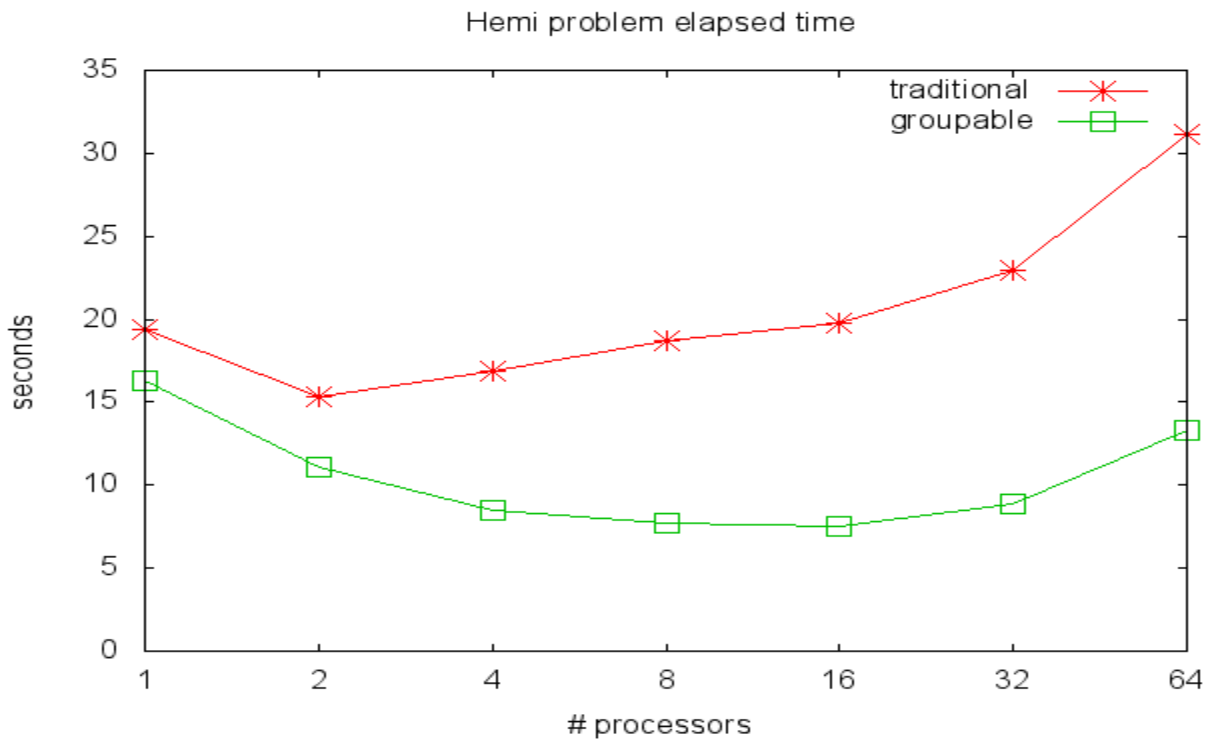

Illustration 12: Hemispherical deep drawing total elapsed time

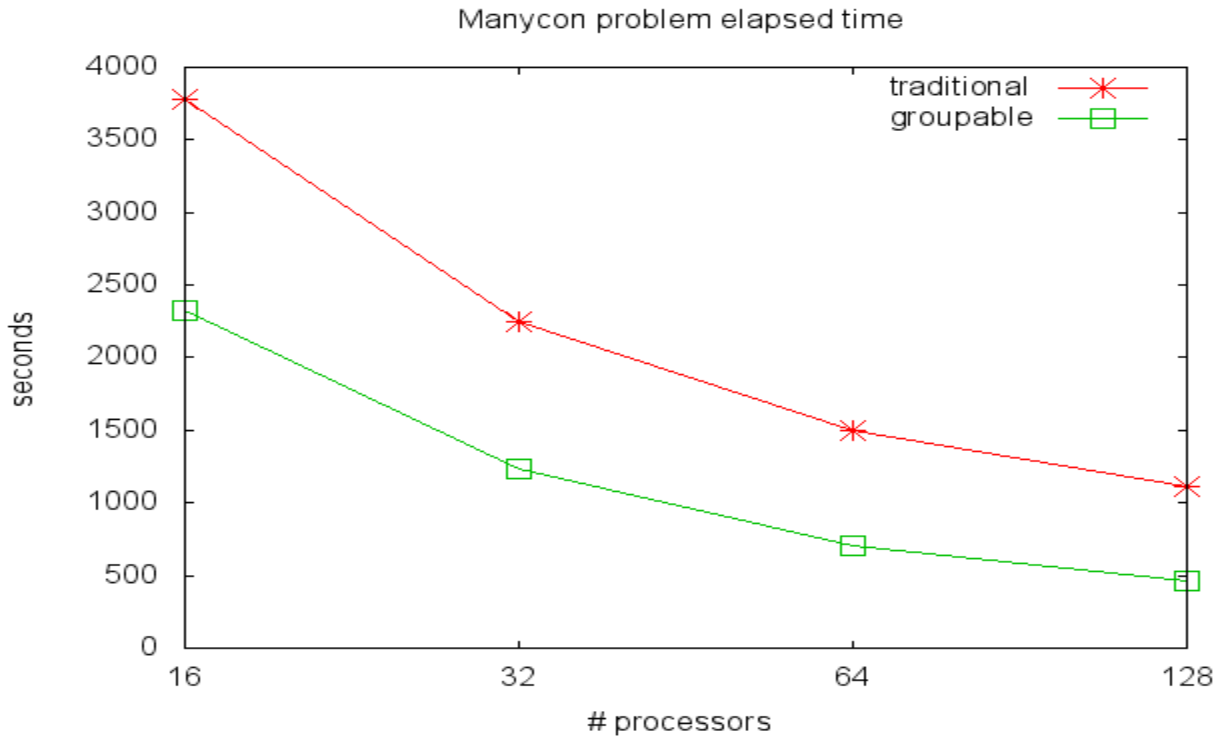

Illustration 13: Manycon problem total elapsed time

the new contact is faster than the old by a factor of 1.6, and on 128 it is faster by a factor of almost 2.4. When increasing from 64 to 128 processors the traditional contact has a speedup of 1.35, while the groupable contact has a speedup of over 1.5

Problem 3: Manufacturing

This model has 8 SURFACE_TO_SURFACE contact interfaces with complex behavior which account for about 60%-80% of the total calculation time. The elapsed times for this model are shown in illustration 14. On 16 processors, the groupable contact gave an improvement of 1.24 over the traditional contact, and on 64 the improvement was a factor of 1.42.

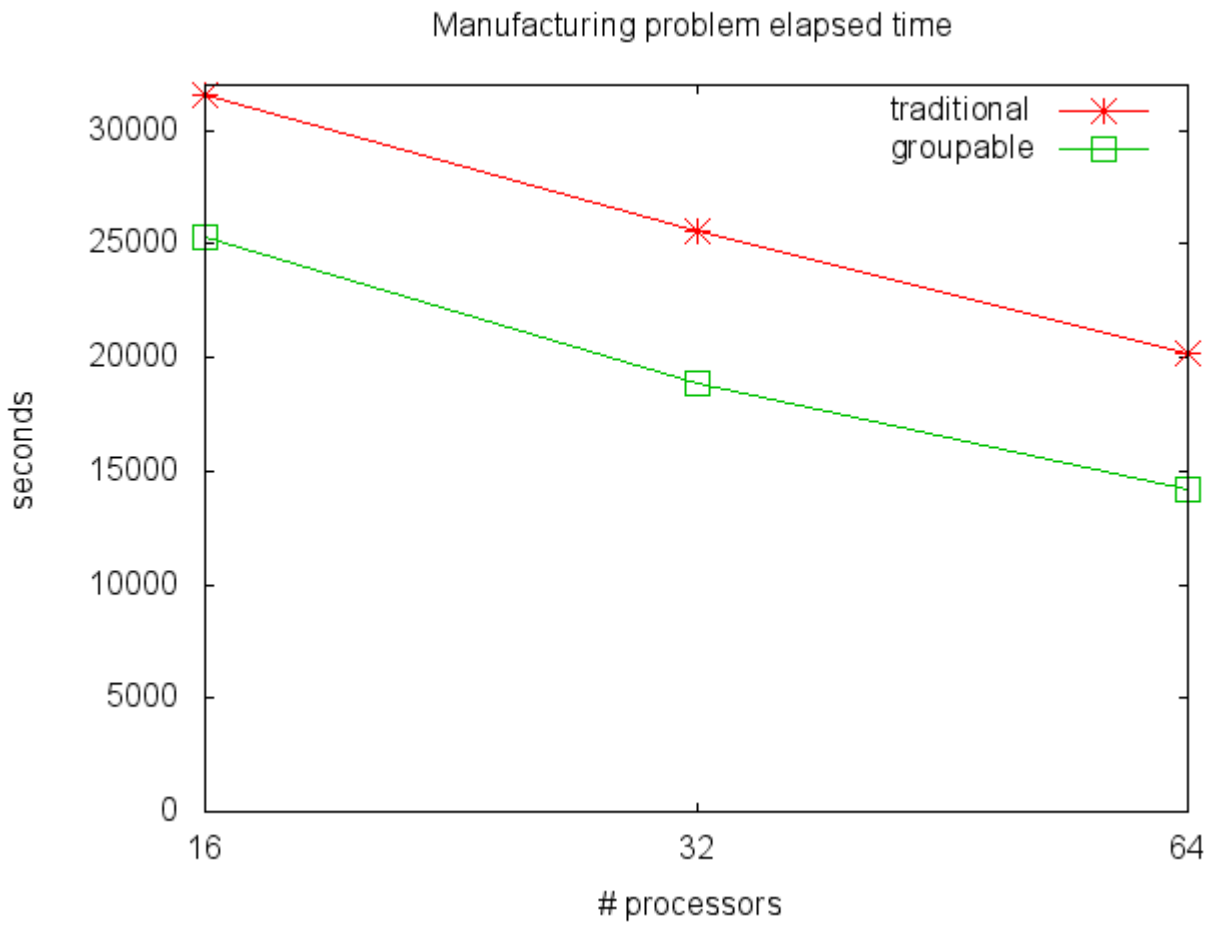

Illustration 14: Manufacturing problem total elapsed time

Application

The “groupable” contact option is available for the following contact types:

```
SINGLE_SURFACE
NODE_TO_SURFACE
SURFACE_TO_SURFACE
ONE_WAY_SURFACE_TO_SURFACE
AUTOMATIC_TIEBREAK
TIED_*
```

There are two ways to control the selection between groupable and traditional contacts, for those contacts for which both algorithms are available. Putting a "1" in the fourth field of the second _MPP optional card can turn on the groupable option on a surface by surface basis, like this:

```
*CONTACT_SINGLE_SURFACE_ID_MPP
$       ID TITLE
       101 Plate to Body
$   ignore    bucket  lcbucket  ns2track  inititer    parmax   ignored    cparm8

$            chksegs     pensf   grpable
&                                     1
```

There is also a way to turn this option on and off globally via the MPP specific parameter file generally known as the pfile. In the "contact" section of the pfile, there is a keyword "groupable" which takes a single numeric argument:

```
contact { groupable N }
```

where N is constructed by adding together these values with their indicated meanings:

1 = turn on the groupable option for all non-tied contacts
2 = turn on the groupable option for all tied contacts
4 = turn off the groupable option for all non-tied contacts
8 = turn off the groupable option for all tied contacts.

For example, adding the line "contact { groupable 9 } would turn on the groupable option for non-tied contacts, and turn off the groupable option for all tied contacts. If the pfile is used, it will override whatever settings are given in the input file. This allows you to easily try turning on or off this option to test it in your specific application.