Datatype Extraction

Extracting declarative descriptions of data layout from MPI packing statements

Fredrik Kjolstad, Torsten Hoefler, Danny Dig and Marc Snir
University of Illinois at Urbana-Champaign, HPC Group

Abstract
Many high performance applications spend considerable time packing data into flat communication buffers. MPI's Derived Datatypes provide an alternative by describing data layout. Datatypes enable runtimes to perform smart optimizations, enabling observed speedups up to 3.8X. Unfortunately, programmers find complex datatypes hard to use and are reluctant to spend time replacing packing code with datatypes. Fortunately, the transformation can be automated, and the programmer can replace packing code with datatypes at the push of a button in a rotatating environment. The transformation allows easy porting of scientific applications to new machines that benefit from datatypes, thereby improving programmer productivity.

Example
C99 packing code that copies the two first columns of a matrix of doubles into a send buffer

```
double buffer[2*N];
for(int i=0; i< N; i++) {
    buffer[2*i] = M[i][0];
    buffer[2*i+1] = M[i][1];
}
MPI_Send(buffer, 2*N, MPI_DOUBLE, 0, tag, COMM);
```

Intermediate code

```
MPI_DataType struct1_t;
MPI_DATA_TRUNCATE(struct1_t);
MPI_STRUCT_INITIALIZER(struct1_t, 2*N, 1, 0, M[0][0], M[0][1],

```

Packaging Intermediate Representation

**PackingGroup** is one of:

- **Packing Statement**
  - Any statement that copies data into the send buffer
- **Packing Sequence**
  - A sequence of packing groups
- **Packing Loop**
  - A loop containing one nested packing group

**Datatype** is one of the MPI datatype classes:

- **Struct**: Bytewise indexed heterogeneous data
- **Hindexed**: Byte indexed homogeneous data
- **Indexed**: Indexed homogeneous data
- **Indexed_block**: Indexed with fixed block sizes
- **Vector**: Strided data with bytewise strides
- **Contiguous**: Strided data with element strides
- **Contiguous**: Contiguous elements
- **Basic**: MPI_INT, MPI_FLOAT, MPI_DOUBLE, ...

Datatypes as Gather/Scatter Transforms

A datatype, when used in send operation, describes a gather operation from arbitrary data layouts to contiguous data (appears descriptive scatters):

```
struct { M[0][0], M[0][1]...M[N-1][0], M[N-1][1] };
```

```
```

Specialization Steps

**Struct to Hindexed**

- Determine whether all elements are of the same type.

**Hindexed to Vector**

- Determine whether the current packing statement is a contiguous access to the data structure (the index expression is fixed).
- If the index expression is fixed, then the sub-intensive part is moved from the original packing code.
- If the index expression is fixed, then the sub-intensive part is moved from the original packing code.

**Contiguous to Vector**

- Determine whether the vector size is a multiple of the subtype (bytewise).

Conclusions

- Datatypes free up runtime to optimize data marshaling, thus alleviating the application programmer from this burden [2].
- Non-indexed datatypes describe data layouts succinctly, thus improving code readability.
- Datatypes allow advanced runtime to avoid buffer copies, by transmitting data directly from/to application data structures (zero-copy) [3].
- Speedups up to 2.8X and 18%, from replacing packing code with datatypes have been observed in a Fast Fourier Transform code and a Conjugate Gradient Solver (MLC) respectively [1].

Our technique is currently being implemented as a reactoring tool in the Eclipse CDT IDE, but can also be implemented in an MPI-aware compiler.

References