WaveSimC

0.8

Generated by Doxygen 1.9.6

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 Np

Namespaces

- namespace np

  Custom implementation of numpy in C++.

Functions

- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > operator* (boost::multi_array< T, ND > const &lhs, boost::multi_array< T, ND > const &rhs)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > operator+ (boost::multi_array< T, ND > const &lhs, boost::multi_array< T, ND > const &rhs)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > operator- (boost::multi_array< T, ND > const &lhs, boost::multi_array< T, ND > const &rhs)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > operator/ (boost::multi_array< T, ND > const &lhs, boost::multi_array< T, ND > const &rhs)

4.1.1 Detailed Description

4.1.2 Function Documentation

4.1.2.1 operator∗()

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > operator* (
              boost::multi_array< T, ND > const & lhs,
              boost::multi_array< T, ND > const & rhs )  [inline]
```

Definition at line 365 of file np.hpp.

```
00366 {
00367     std::function<T(T, T)> func = std::multiplies<T>();
00368     return np::element_wise_duo_apply(lhs, rhs, func);
00369 }
```

4.1.2.2 operator+()

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > operator+ (
              boost::multi_array< T, ND > const & lhs,
              boost::multi_array< T, ND > const & rhs )
```

Definition at line 371 of file np.hpp.

```
00372 {
00373     std::function<T(T, T)> func = std::plus<T>();
00374     return np::element_wise_duo_apply(lhs, rhs, func);
00375 }
```

4.1.2.3 operator-()

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > operator- (
              boost::multi_array< T, ND > const & lhs,
              boost::multi_array< T, ND > const & rhs )
```

Definition at line 377 of file np.hpp.

```
00378 {
00379     std::function<T(T, T)> func = std::minus<T>();
00380     return np::element_wise_duo_apply(lhs, rhs, func);
00381 }
```

4.1.2.4 operator/()

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > operator/ (
              boost::multi_array< T, ND > const & lhs,
              boost::multi_array< T, ND > const & rhs )
```

Definition at line 383 of file np.hpp.

```
00384 {
00385     std::function<T(T, T)> func = std::divides<T>();
00386     return np::element_wise_duo_apply(lhs, rhs, func);
00387 }
```

Chapter 5

Namespace Documentation

5.1 np Namespace Reference

Custom implementation of numpy in C++.

Typedefs

- typedef double ndArrayValue

Enumerations

- enum **indexing** { **xy** , **ij** }

Functions

- template<std::size_t ND>
  boost::multi_array< ndArrayValue, ND >::index getIndex (const boost::multi_array< ndArrayValue, ND > &m, const ndArrayValue *requestedElement, const unsigned short int direction)

  Gets the index of one element in a multi_array in one axis.
- template<std::size_t ND>
  boost::array< typename boost::multi_array< ndArrayValue, ND >::index, ND > getIndexArray (const boost::multi_array< ndArrayValue, ND > &m, const ndArrayValue *requestedElement)

  Gets the index of one element in a multi_array.
- template<typename Array , typename Element , typename Functor >
  void for_each (const boost::type< Element > &type_dispatch, Array A, Functor &xform)
- template<typename Element , typename Functor >
  void for_each (const boost::type< Element > &, Element &Val, Functor &xform)

  Function to apply a function to all elements of a multi_array.
- template<typename Element , typename Iterator , typename Functor >
  void for_each (const boost::type< Element > &type_dispatch, Iterator begin, Iterator end, Functor &xform)

  Function to apply a function to all elements of a multi_array.
- template<typename Array , typename Functor >
  void for_each (Array &A, Functor xform)

- template<long unsigned int ND>
  constexpr std::vector< boost::multi_array< double, ND > > gradient (boost::multi_array< double, ND > inArray, std::initializer_list< double > args)
- boost::multi_array< double, 1 > linspace (double start, double stop, long unsigned int num)

  Implements the numpy linspace function.
- boost::multi_array< double, 1 > zeros (long unsigned int num)
- template<long unsigned int ND>
  std::vector< boost::multi_array< double, ND > > meshgrid (const boost::multi_array< double, 1 >(&cinput)[ND], bool sparsing=false, indexing indexing_type=xy)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > element_wise_apply (const boost::multi_array< T, ND > &input_array, std::function< T(T)> func)

  Cretes a new array and fills it with the values of the result of the function called on the input array element-wise.
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > sqrt (const boost::multi_array< T, ND > &input_array)
- template<class T >
  T sqrt (const T input)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > exp (const boost::multi_array< T, ND > &input_array)
- template<class T >
  T exp (const T input)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > log (const boost::multi_array< T, ND > &input_array)
- template<class T >
  T log (const T input)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > pow (const boost::multi_array< T, ND > &input_array, const T exponent)
- template<class T >
  T pow (const T input, const T exponent)
- template<class T , long unsigned int ND>
  boost::multi_array< T, ND > element_wise_duo_apply (boost::multi_array< T, ND > const &lhs, boost::multi_array< T, ND > const &rhs, std::function< T(T, T)> func)

5.1.1 Detailed Description

Custom implementation of numpy in C++.

5.1.2 Typedef Documentation

5.1.2.1 ndArrayValue

```
typedef double np::ndArrayValue
```

Definition at line 21 of file np.hpp.

5.1.3 Enumeration Type Documentation

5.1.3.1 indexing

```
enum np::indexing
```

Definition at line 183 of file np.hpp.

```
00184     {
00185         xy,
00186         ij
00187     };
```

5.1.4 Function Documentation

5.1.4.1 element_wise_apply()

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > np::element_wise_apply (
            const boost::multi_array< T, ND > & input_array,
            std::function< T(T)> func )  [inline]
```

Cretes a new array and fills it with the values of the result of the function called on the input array element-wise.

Definition at line 254 of file np.hpp.

```
00255     {
00256 
00257         // Create output array copying extents
00258         using arrayIndex = boost::multi_array<double, ND>::index;
00259         using ndIndexArray = boost::array<arrayIndex, ND>;
00260         boost::detail::multi_array::extent_gen<ND> output_extents;
00261         std::vector<size_t> shape_list;
00262         for (std::size_t i = 0; i < ND; i++)
00263         {
00264             shape_list.push_back(input_array.shape()[i]);
00265         }
00266         std::copy(shape_list.begin(), shape_list.end(), output_extents.ranges_.begin());
00267         boost::multi_array<T, ND> output_array(output_extents);
00268 
00269         // Looping through the elements of the output array
00270         const T *p = input_array.data();
00271         ndIndexArray index;
00272         for (std::size_t i = 0; i < input_array.num_elements(); i++)
00273         {
00274             index = getIndexArray(input_array, p);
00275             output_array(index) = func(input_array(index));
00276             ++p;
00277         }
00278         return output_array;
00279     }
```

5.1.4.2 element_wise_duo_apply()

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > np::element_wise_duo_apply (
            boost::multi_array< T, ND > const & lhs,
            boost::multi_array< T, ND > const & rhs,
            std::function< T(T, T)> func )
```

Creates a new array in which the value at each index is the the result of the input function applied to an element of the left hand side array and one on the righ hand side array in the same index Outputs a copy of the result

Definition at line 335 of file np.hpp.

```
00336     {
00337         // Create output array copying extents
00338         using arrayIndex = boost::multi_array<double, ND>::index;
00339         using ndIndexArray = boost::array<arrayIndex, ND>;
00340         boost::detail::multi_array::extent_gen<ND> output_extents;
00341         std::vector<size_t> shape_list;
00342         for (std::size_t i = 0; i < ND; i++)
00343         {
00344             shape_list.push_back(lhs.shape()[i]);
00345         }
00346         std::copy(shape_list.begin(), shape_list.end(), output_extents.ranges_.begin());
00347         boost::multi_array<T, ND> output_array(output_extents);
00348 
00349         // Looping through the elements of the output array
00350         const T *p = lhs.data();
00351         ndIndexArray index;
00352         for (std::size_t i = 0; i < lhs.num_elements(); i++)
00353         {
00354             index = getIndexArray(lhs, p);
00355             output_array(index) = func(lhs(index), rhs(index));
00356             ++p;
00357         }
00358         return output_array;
00359     }
```

5.1.4.3 exp() [1/2]

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > np::exp (
             const boost::multi_array< T, ND > & input_array )  [inline]
```

Definition at line 296 of file np.hpp.

```
00297     {
00298         std::function<T(T)> func = (T(*)(T))std::exp;
00299         return element_wise_apply(input_array, func);
00300     }
```

5.1.4.4 exp() [2/2]

```
template<class T >
T np::exp (
             const T input )  [inline]
```

Definition at line 302 of file np.hpp.

```
00303     {
00304         return std::exp(input);
00305     }
```

5.1.4.5 for_each() [1/4]

```
template<typename Array , typename Functor >
void np::for_each (
             Array & A,
             Functor xform )  [inline]
```

Function to apply a function to all elements of a multi_array Simple overload

Definition at line 79 of file np.hpp.

```
00080     {
00081         // Dispatch to the proper function
00082         for_each(boost::type<typename Array::element>(), A.begin(), A.end(), xform);
00083     }
```

5.1.4.6 for_each() [2/4]

```
template<typename Element , typename Functor >
void np::for_each (
             const boost::type< Element > & ,
             Element & Val,
             Functor & xform )  [inline]
```

Function to apply a function to all elements of a multi_array.

Definition at line 58 of file np.hpp.

```
00059     {
00060         Val = xform(Val);
00061     }
```

5.1.4.7 for_each() [3/4]

```
template<typename Array , typename Element , typename Functor >
void np::for_each (
             const boost::type< Element > & type_dispatch,
             Array A,
             Functor & xform )  [inline]
```

Function to apply a function to all elements of a multi_array Simple overload

Definition at line 50 of file np.hpp.

```
00052     {
00053         for_each(type_dispatch, A.begin(), A.end(), xform);
00054     }
```

5.1.4.8 for_each() [4/4]

```
template<typename Element , typename Iterator , typename Functor >
void np::for_each (
             const boost::type< Element > & type_dispatch,
             Iterator begin,
             Iterator end,
             Functor & xform )  [inline]
```

Function to apply a function to all elements of a multi_array.

Definition at line 65 of file np.hpp.

```
00068     {
00069         while (begin != end)
00070         {
00071             for_each(type_dispatch, *begin, xform);
00072             ++begin;
00073         }
00074     }
```

5.1.4.9 getIndex()

```
template<std::size_t ND>
boost::multi_array< ndArrayValue, ND >::index np::getIndex (
            const boost::multi_array< ndArrayValue, ND > & m,
            const ndArrayValue * requestedElement,
            const unsigned short int direction )  [inline]
```

Gets the index of one element in a multi_array in one axis.

Definition at line 26 of file np.hpp.

```
00027     {
00028         int offset = requestedElement - m.origin();
00029         return (offset / m.strides()[direction] % m.shape()[direction] + m.index_bases()[direction]);
00030     }
```

5.1.4.10 getIndexArray()

```
template<std::size_t ND>
boost::array< typename boost::multi_array< ndArrayValue, ND >::index, ND > np::getIndexArray
(
            const boost::multi_array< ndArrayValue, ND > & m,
            const ndArrayValue * requestedElement )  [inline]
```

Gets the index of one element in a multi_array.

Definition at line 35 of file np.hpp.

```
00036     {
00037         using indexType = boost::multi_array<ndArrayValue, ND>::index;
00038         boost::array<indexType, ND> _index;
00039         for (unsigned int dir = 0; dir < ND; dir++)
00040         {
00041             _index[dir] = getIndex(m, requestedElement, dir);
00042         }
00043
00044         return _index;
00045     }
```

5.1.4.11 gradient()

```
template<long unsigned int ND>
constexpr std::vector< boost::multi_array< double, ND > > np::gradient (
            boost::multi_array< double, ND > inArray,
            std::initializer_list< double > args )  [inline], [constexpr]
```

Takes the gradient of a n-dimensional multi_array Todo: Actually implement the gradient calculation template <long unsigned int ND, typename... Args>

Definition at line 89 of file np.hpp.

```
00090     {
00091         // static_assert(args.size() == ND, "Number of arguments must match the number of dimensions
      of the array");
00092         using arrayIndex = boost::multi_array<double, ND>::index;
00093
00094         using ndIndexArray = boost::array<arrayIndex, ND>;
00095
00096         // constexpr std::size_t n = sizeof...(Args);
00097         std::size_t n = args.size();
00098         // std::tuple<Args...> store(args...);
```

```
00099         std::vector<double> arg_vector = args;
00100         boost::multi_array<double, ND> my_array;
00101         std::vector<boost::multi_array<double, ND» output_arrays;
00102         for (std::size_t i = 0; i < n; i++)
00103         {
00104             boost::multi_array<double, ND> dfdh = inArray;
00105             output_arrays.push_back(dfdh);
00106         }
00107 
00108         ndArrayValue *p = inArray.data();
00109         ndIndexArray index;
00110         for (std::size_t i = 0; i < inArray.num_elements(); i++)
00111         {
00112             index = getIndexArray(inArray, p);
00113             /*
00114             std::cout « "Index: ";
00115             for (std::size_t j = 0; j < n; j++)
00116             {
00117                 std::cout « index[j] « " ";
00118             }
00119             std::cout « "\n";
00120             */
00121             // Calculating the gradient now
00122             // j is the axis/dimension
00123             for (std::size_t j = 0; j < n; j++)
00124             {
00125                 ndIndexArray index_high = index;
00126                 double dh_high;
00127                 if ((long unsigned int)index_high[j] < inArray.shape()[j] - 1)
00128                 {
00129                     index_high[j] += 1;
00130                     dh_high = arg_vector[j];
00131                 }
00132                 else
00133                 {
00134                     dh_high = 0;
00135                 }
00136                 ndIndexArray index_low = index;
00137                 double dh_low;
00138                 if (index_low[j] > 0)
00139                 {
00140                     index_low[j] -= 1;
00141                     dh_low = arg_vector[j];
00142                 }
00143                 else
00144                 {
00145                     dh_low = 0;
00146                 }
00147 
00148                 double dh = dh_high + dh_low;
00149                 double gradient = (inArray(index_high) - inArray(index_low)) / dh;
00150                 // std::cout « gradient « "\n";
00151                 output_arrays[j](index) = gradient;
00152             }
00153             // std::cout « " value = " « inArray(index) « "  check = " « *p « std::endl;
00154             ++p;
00155         }
00156         return output_arrays;
00157     }
```

5.1.4.12 linspace()

```
boost::multi_array< double, 1 > np::linspace (
            double start,
            double stop,
            long unsigned int num )  [inline]
```

Implements the numpy linspace function.

Definition at line 160 of file np.hpp.

```
00161     {
00162         double step = (stop - start) / (num - 1);
00163         boost::multi_array<double, 1> output(boost::extents[num]);
00164         for (std::size_t i = 0; i < num; i++)
00165         {
00166             output[i] = start + i * step;
00167         }
00168         return output;
00169     }
```

5.1.4.13 log() [1/2]

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > np::log (
            const boost::multi_array< T, ND > & input_array )  [inline]
```

Definition at line 308 of file np.hpp.

```
00309     {
00310         std::function<T(T)> func = std::log<T>();
00311         return element_wise_apply(input_array, func);
00312     }
```

5.1.4.14 log() [2/2]

```
template<class T >
T np::log (
            const T input )  [inline]
```

Definition at line 314 of file np.hpp.

```
00315     {
00316         return std::log(input);
00317     }
```

5.1.4.15 meshgrid()

```
template<long unsigned int ND>
std::vector< boost::multi_array< double, ND > > np::meshgrid (
            const boost::multi_array< double, 1 >(&) cinput[ND],
            bool sparsing = false,
            indexing indexing_type = xy )  [inline]
```

Implementation of meshgrid TODO: Implement sparsing=true If the indexing type is xx, then reverse the order of the first two elements of ci if the number of dimensions is 2 or 3 In accordance with the numpy implementation

Definition at line 195 of file np.hpp.

```
00196     {
00197         using arrayIndex = boost::multi_array<double, ND>::index;
00198         using ndIndexArray = boost::array<arrayIndex, ND>;
00199         std::vector<boost::multi_array<double, ND» output_arrays;
00200         boost::multi_array<double, 1> ci[ND];
00201         // Copy elements of cinput to ci, do the proper inversions
00202         for (std::size_t i = 0; i < ND; i++)
00203         {
00204             std::size_t source = i;
00205             if (indexing_type == xy && (ND == 3 || ND == 2))
00206             {
00207                 switch (i)
00208                 {
00209                 case 0:
00210                     source = 1;
00211                     break;
00212                 case 1:
00213                     source = 0;
00214                     break;
00215                 default:
00216                     break;
00217                 }
00218             }
00219             ci[i] = boost::multi_array<double, 1>();
00220             ci[i].resize(boost::extents[cinput[source].num_elements()]);
00221             ci[i] = cinput[source];
00222         }
```

```
00223         // Deducing the extents of the N-Dimensional output
00224         boost::detail::multi_array::extent_gen<ND> output_extents;
00225         std::vector<size_t> shape_list;
00226         for (std::size_t i = 0; i < ND; i++)
00227         {
00228             shape_list.push_back(ci[i].shape()[0]);
00229         }
00230         std::copy(shape_list.begin(), shape_list.end(), output_extents.ranges_.begin());
00231 
00232         // Creating the output arrays
00233         for (std::size_t i = 0; i < ND; i++)
00234         {
00235             boost::multi_array<double, ND> output_array(output_extents);
00236             ndArrayValue *p = output_array.data();
00237             ndIndexArray index;
00238             // Looping through the elements of the output array
00239             for (std::size_t j = 0; j < output_array.num_elements(); j++)
00240             {
00241                 index = getIndexArray(output_array, p);
00242                 boost::multi_array<double, 1>::index index_1d;
00243                 index_1d = index[i];
00244                 output_array(index) = ci[i][index_1d];
00245                 ++p;
00246             }
00247             output_arrays.push_back(output_array);
00248         }
00249         return output_arrays;
00250     }
```

5.1.4.16 pow() [1/2]

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > np::pow (
             const boost::multi_array< T, ND > & input_array,
             const T exponent )  [inline]
```

Definition at line 319 of file np.hpp.

```
00320     {
00321         std::function<T(T)> pow_func = [exponent](T input)
00322         { return std::pow(input, exponent); };
00323         return element_wise_apply(input_array, pow_func);
00324     }
```

5.1.4.17 pow() [2/2]

```
template<class T >
T np::pow (
             const T input,
             const T exponent )  [inline]
```

Definition at line 326 of file np.hpp.

```
00327     {
00328         return std::pow(input, exponent);
00329     }
```

5.1.4.18 sqrt() [1/2]

```
template<class T , long unsigned int ND>
boost::multi_array< T, ND > np::sqrt (
            const boost::multi_array< T, ND > & input_array )  [inline]
```

Definition at line 284 of file np.hpp.

```
00285     {
00286         std::function<T(T)> func = (T(*)(T))std::sqrt;
00287         return element_wise_apply(input_array, func);
00288     }
```

5.1.4.19 sqrt() [2/2]

```
template<class T >
T np::sqrt (
            const T input )  [inline]
```

Definition at line 290 of file np.hpp.

```
00291     {
00292         return std::sqrt(input);
00293     }
```

5.1.4.20 zeros()

```
boost::multi_array< double, 1 > np::zeros (
            long unsigned int num )  [inline]
```

Implements the numpy zeros function Todo: make it work for any number of dimensions

Definition at line 173 of file np.hpp.

```
00174     {
00175         boost::multi_array<double, 1> output(boost::extents[num]);
00176         for (std::size_t i = 0; i < num; i++)
00177         {
00178             output[i] = 0;
00179         }
00180         return output;
00181     }
```

Chapter 6

File Documentation

6.1 CMakeCCompilerId.c

```
00001 #ifdef __cplusplus
00002 # error "A C++ compiler has been selected for C."
00003 #endif
00004
00005 #if defined(__18CXX)
00006 # define ID_VOID_MAIN
00007 #endif
00008 #if defined(__CLASSIC_C__)
00009 /* cv-qualifiers did not exist in K&R C */
00010 # define const
00011 # define volatile
00012 #endif
00013
00014
00015 /* Version number components: V=Version, R=Revision, P=Patch
00016    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00017
00018 #if defined(__INTEL_COMPILER) || defined(__ICC)
00019 # define COMPILER_ID "Intel"
00020 # if defined(_MSC_VER)
00021 #  define SIMULATE_ID "MSVC"
00022 # endif
00023 # if defined(__GNUC__)
00024 #  define SIMULATE_ID "GNU"
00025 # endif
00026   /* __INTEL_COMPILER = VRP */
00027 # define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00028 # define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00029 # if defined(__INTEL_COMPILER_UPDATE)
00030 #  define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00031 # else
00032 #  define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00033 # endif
00034 # if defined(__INTEL_COMPILER_BUILD_DATE)
00035   /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00036 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00037 # endif
00038 # if defined(_MSC_VER)
00039    /* _MSC_VER = VVRR */
00040 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00041 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00042 # endif
00043 # if defined(__GNUC__)
00044 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00045 # elif defined(__GNUG__)
00046 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00047 # endif
00048 # if defined(__GNUC_MINOR__)
00049 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00050 # endif
00051 # if defined(__GNUC_PATCHLEVEL__)
00052 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00053 # endif
00054
00055 #elif defined(__PATHCC__)
00056 # define COMPILER_ID "PathScale"
00057 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00058 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
```

```
00059 # if defined(__PATHCC_PATCHLEVEL__)
00060 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00061 # endif
00062
00063 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00064 # define COMPILER_ID "Embarcadero"
00065 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00066 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00067 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00068
00069 #elif defined(__BORLANDC__)
00070 # define COMPILER_ID "Borland"
00071   /* __BORLANDC__ = 0xVRR */
00072 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00073 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00074
00075 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00076 # define COMPILER_ID "Watcom"
00077    /* __WATCOMC__ = VVRR */
00078 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00079 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00080 # if (__WATCOMC__ % 10) > 0
00081 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00082 # endif
00083
00084 #elif defined(__WATCOMC__)
00085 # define COMPILER_ID "OpenWatcom"
00086    /* __WATCOMC__ = VVRP + 1100 */
00087 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00088 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00089 # if (__WATCOMC__ % 10) > 0
00090 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00091 # endif
00092
00093 #elif defined(__SUNPRO_C)
00094 # define COMPILER_ID "SunPro"
00095 # if __SUNPRO_C >= 0x5100
00096    /* __SUNPRO_C = 0xVRRP */
00097 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C»12)
00098 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C»4 & 0xFF)
00099 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00100 # else
00101    /* __SUNPRO_CC = 0xVRP */
00102 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C»8)
00103 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C»4 & 0xF)
00104 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00105 # endif
00106
00107 #elif defined(__HP_cc)
00108 # define COMPILER_ID "HP"
00109   /* __HP_cc = VVRRPP */
00110 # define COMPILER_VERSION_MAJOR DEC(__HP_cc/10000)
00111 # define COMPILER_VERSION_MINOR DEC(__HP_cc/100 % 100)
00112 # define COMPILER_VERSION_PATCH DEC(__HP_cc     % 100)
00113
00114 #elif defined(__DECC)
00115 # define COMPILER_ID "Compaq"
00116   /* __DECC_VER = VVRRTPPPP */
00117 # define COMPILER_VERSION_MAJOR DEC(__DECC_VER/10000000)
00118 # define COMPILER_VERSION_MINOR DEC(__DECC_VER/100000  % 100)
00119 # define COMPILER_VERSION_PATCH DEC(__DECC_VER         % 10000)
00120
00121 #elif defined(__IBMC__) && defined(__COMPILER_VER__)
00122 # define COMPILER_ID "zOS"
00123   /* __IBMC__ = VRP */
00124 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00125 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00126 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00127
00128 #elif defined(__ibmxl__) && defined(__clang__)
00129 # define COMPILER_ID "XLClang"
00130 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00131 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00132 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00133 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00134
00135
00136 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ >= 800
00137 # define COMPILER_ID "XL"
00138   /* __IBMC__ = VRP */
00139 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00140 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00141 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00142
00143 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ < 800
00144 # define COMPILER_ID "VisualAge"
00145   /* __IBMC__ = VRP */
```

```
00146 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00147 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00148 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00149
00150 #elif defined(__PGI)
00151 # define COMPILER_ID "PGI"
00152 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00153 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00154 # if defined(__PGIC_PATCHLEVEL__)
00155 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00156 # endif
00157
00158 #elif defined(_CRAYC)
00159 # define COMPILER_ID "Cray"
00160 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00161 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00162
00163 #elif defined(__TI_COMPILER_VERSION__)
00164 # define COMPILER_ID "TI"
00165   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00166 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00167 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00168 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00169
00170 #elif defined(__FUJITSU) || defined(__FCC_VERSION) || defined(__fcc_version)
00171 # define COMPILER_ID "Fujitsu"
00172
00173 #elif defined(__ghs__)
00174 # define COMPILER_ID "GHS"
00175 /* __GHS_VERSION_NUMBER = VVVVRP */
00176 # ifdef __GHS_VERSION_NUMBER
00177 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00178 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00180 # endif
00181
00182 #elif defined(__TINYC__)
00183 # define COMPILER_ID "TinyCC"
00184
00185 #elif defined(__BCC__)
00186 # define COMPILER_ID "Bruce"
00187
00188 #elif defined(__SCO_VERSION__)
00189 # define COMPILER_ID "SCO"
00190
00191 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00192 # define COMPILER_ID "ARMCC"
00193 #if __ARMCC_VERSION >= 1000000
00194   /* __ARMCC_VERSION = VRRPPPP */
00195   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00196   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00197   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00198 #else
00199   /* __ARMCC_VERSION = VRPPPP */
00200   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00201   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00202   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00203 #endif
00204
00205
00206 #elif defined(__clang__) && defined(__apple_build_version__)
00207 # define COMPILER_ID "AppleClang"
00208 # if defined(_MSC_VER)
00209 #  define SIMULATE_ID "MSVC"
00210 # endif
00211 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00212 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00213 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00214 # if defined(_MSC_VER)
00215    /* _MSC_VER = VVRR */
00216 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00217 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00218 # endif
00219 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00220
00221 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00222 # define COMPILER_ID "ARMClang"
00223   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00224   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00225   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION     % 10000)
00226 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00227
00228 #elif defined(__clang__)
00229 # define COMPILER_ID "Clang"
00230 # if defined(_MSC_VER)
00231 #  define SIMULATE_ID "MSVC"
00232 # endif
```

```
00233 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00234 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00235 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00236 # if defined(_MSC_VER)
00237   /* _MSC_VER = VVRR */
00238 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00239 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00240 # endif
00241
00242 #elif defined(__GNUC__)
00243 # define COMPILER_ID "GNU"
00244 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00245 # if defined(__GNUC_MINOR__)
00246 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00247 # endif
00248 # if defined(__GNUC_PATCHLEVEL__)
00249 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00250 # endif
00251
00252 #elif defined(_MSC_VER)
00253 # define COMPILER_ID "MSVC"
00254   /* _MSC_VER = VVRR */
00255 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00256 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00257 # if defined(_MSC_FULL_VER)
00258 #  if _MSC_VER >= 1400
00259     /* _MSC_FULL_VER = VVRRPPPPP */
00260 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00261 #  else
00262     /* _MSC_FULL_VER = VVRRPPPP */
00263 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00264 #  endif
00265 # endif
00266 # if defined(_MSC_BUILD)
00267 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00268 # endif
00269
00270 #elif defined(__VISUALDSPVERSION__) || defined(__ADSPBLACKFIN__) || defined(__ADSPTS__) ||
      defined(__ADSP21000__)
00271 # define COMPILER_ID "ADSP"
00272 #if defined(__VISUALDSPVERSION__)
00273   /* __VISUALDSPVERSION__ = 0xVVRRPP00 */
00274 # define COMPILER_VERSION_MAJOR HEX(__VISUALDSPVERSION__»24)
00275 # define COMPILER_VERSION_MINOR HEX(__VISUALDSPVERSION__»16 & 0xFF)
00276 # define COMPILER_VERSION_PATCH HEX(__VISUALDSPVERSION__»8  & 0xFF)
00277 #endif
00278
00279 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00280 # define COMPILER_ID "IAR"
00281 # if defined(__VER__) && defined(__ICCARM__)
00282 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00283 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00284 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00285 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00286 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
      defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
      defined(__ICC8051__))
00287 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00288 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00289 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00290 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00291 # endif
00292
00293 #elif defined(__SDCC_VERSION_MAJOR) || defined(SDCC)
00294 # define COMPILER_ID "SDCC"
00295 # if defined(__SDCC_VERSION_MAJOR)
00296 #  define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00297 #  define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00298 #  define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00299 # else
00300   /* SDCC = VRP */
00301 #  define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00302 #  define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00303 #  define COMPILER_VERSION_PATCH DEC(SDCC    % 10)
00304 # endif
00305
00306
00307 /* These compilers are either not known or too old to define an
00308   identification macro.  Try to identify the platform and guess that
00309   it is the native compiler.  */
00310 #elif defined(__hpux) || defined(__hpua)
00311 # define COMPILER_ID "HP"
00312
00313 #else /* unknown compiler */
00314 # define COMPILER_ID ""
00315 #endif
00316
```

```
00317 /* Construct the string literal in pieces to prevent the source from
00318    getting matched.  Store it in a pointer rather than an array
00319    because some compilers will just produce instructions to fill the
00320    array rather than assigning a pointer to a static array.  */
00321 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00322 #ifdef SIMULATE_ID
00323 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00324 #endif
00325
00326 #ifdef __QNXNTO__
00327 char const* qnxnto = "INFO" ":" "qnxnto[]";
00328 #endif
00329
00330 #if defined(__CRAYXE) || defined(__CRAYXC)
00331 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00332 #endif
00333
00334 #define STRINGIFY_HELPER(X) #X
00335 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00336
00337 /* Identify known platforms by name.  */
00338 #if defined(__linux) || defined(__linux__) || defined(linux)
00339 # define PLATFORM_ID "Linux"
00340
00341 #elif defined(__CYGWIN__)
00342 # define PLATFORM_ID "Cygwin"
00343
00344 #elif defined(__MINGW32__)
00345 # define PLATFORM_ID "MinGW"
00346
00347 #elif defined(__APPLE__)
00348 # define PLATFORM_ID "Darwin"
00349
00350 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00351 # define PLATFORM_ID "Windows"
00352
00353 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00354 # define PLATFORM_ID "FreeBSD"
00355
00356 #elif defined(__NetBSD__) || defined(__NetBSD)
00357 # define PLATFORM_ID "NetBSD"
00358
00359 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00360 # define PLATFORM_ID "OpenBSD"
00361
00362 #elif defined(__sun) || defined(sun)
00363 # define PLATFORM_ID "SunOS"
00364
00365 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00366 # define PLATFORM_ID "AIX"
00367
00368 #elif defined(__hpux) || defined(__hpux__)
00369 # define PLATFORM_ID "HP-UX"
00370
00371 #elif defined(__HAIKU__)
00372 # define PLATFORM_ID "Haiku"
00373
00374 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00375 # define PLATFORM_ID "BeOS"
00376
00377 #elif defined(__QNX__) || defined(__QNXNTO__)
00378 # define PLATFORM_ID "QNX"
00379
00380 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00381 # define PLATFORM_ID "Tru64"
00382
00383 #elif defined(__riscos) || defined(__riscos__)
00384 # define PLATFORM_ID "RISCos"
00385
00386 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00387 # define PLATFORM_ID "SINIX"
00388
00389 #elif defined(__UNIX_SV__)
00390 # define PLATFORM_ID "UNIX_SV"
00391
00392 #elif defined(__bsdos__)
00393 # define PLATFORM_ID "BSDOS"
00394
00395 #elif defined(_MPRAS) || defined(MPRAS)
00396 # define PLATFORM_ID "MP-RAS"
00397
00398 #elif defined(__osf) || defined(__osf__)
00399 # define PLATFORM_ID "OSF1"
00400
00401 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00402 # define PLATFORM_ID "SCO_SV"
00403
```

```
00404 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00405 # define PLATFORM_ID "ULTRIX"
00406
00407 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00408 # define PLATFORM_ID "Xenix"
00409
00410 #elif defined(__WATCOMC__)
00411 # if defined(__LINUX__)
00412 #  define PLATFORM_ID "Linux"
00413
00414 # elif defined(__DOS__)
00415 #  define PLATFORM_ID "DOS"
00416
00417 # elif defined(__OS2__)
00418 #  define PLATFORM_ID "OS2"
00419
00420 # elif defined(__WINDOWS__)
00421 #  define PLATFORM_ID "Windows3x"
00422
00423 # else /* unknown platform */
00424 #  define PLATFORM_ID
00425 # endif
00426
00427 #elif defined(__INTEGRITY)
00428 # if defined(INT_178B)
00429 #  define PLATFORM_ID "Integrity178"
00430
00431 # else /* regular Integrity */
00432 #  define PLATFORM_ID "Integrity"
00433 # endif
00434
00435 #else /* unknown platform */
00436 # define PLATFORM_ID
00437
00438 #endif
00439
00440 /* For windows compilers MSVC and Intel we can determine
00441    the architecture of the compiler being used.  This is because
00442    the compilers do not have flags that can change the architecture,
00443    but rather depend on which compiler is being used
00444 */
00445 #if defined(_WIN32) && defined(_MSC_VER)
00446 # if defined(_M_IA64)
00447 #  define ARCHITECTURE_ID "IA64"
00448
00449 # elif defined(_M_X64) || defined(_M_AMD64)
00450 #  define ARCHITECTURE_ID "x64"
00451
00452 # elif defined(_M_IX86)
00453 #  define ARCHITECTURE_ID "X86"
00454
00455 # elif defined(_M_ARM64)
00456 #  define ARCHITECTURE_ID "ARM64"
00457
00458 # elif defined(_M_ARM)
00459 #  if _M_ARM == 4
00460 #   define ARCHITECTURE_ID "ARMV4I"
00461 #  elif _M_ARM == 5
00462 #   define ARCHITECTURE_ID "ARMV5I"
00463 #  else
00464 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00465 #  endif
00466
00467 # elif defined(_M_MIPS)
00468 #  define ARCHITECTURE_ID "MIPS"
00469
00470 # elif defined(_M_SH)
00471 #  define ARCHITECTURE_ID "SHx"
00472
00473 # else /* unknown architecture */
00474 #  define ARCHITECTURE_ID ""
00475 # endif
00476
00477 #elif defined(__WATCOMC__)
00478 # if defined(_M_I86)
00479 #  define ARCHITECTURE_ID "I86"
00480
00481 # elif defined(_M_IX86)
00482 #  define ARCHITECTURE_ID "X86"
00483
00484 # else /* unknown architecture */
00485 #  define ARCHITECTURE_ID ""
00486 # endif
00487
00488 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00489 # if defined(__ICCARM__)
00490 #  define ARCHITECTURE_ID "ARM"
```

```
00491
00492 # elif defined(__ICCRX__)
00493 #  define ARCHITECTURE_ID "RX"
00494
00495 # elif defined(__ICCRH850__)
00496 #  define ARCHITECTURE_ID "RH850"
00497
00498 # elif defined(__ICCRL78__)
00499 #  define ARCHITECTURE_ID "RL78"
00500
00501 # elif defined(__ICCRISCV__)
00502 #  define ARCHITECTURE_ID "RISCV"
00503
00504 # elif defined(__ICCAVR__)
00505 #  define ARCHITECTURE_ID "AVR"
00506
00507 # elif defined(__ICC430__)
00508 #  define ARCHITECTURE_ID "MSP430"
00509
00510 # elif defined(__ICCV850__)
00511 #  define ARCHITECTURE_ID "V850"
00512
00513 # elif defined(__ICC8051__)
00514 #  define ARCHITECTURE_ID "8051"
00515
00516 # else /* unknown architecture */
00517 #  define ARCHITECTURE_ID ""
00518 # endif
00519
00520 #elif defined(__ghs__)
00521 # if defined(__PPC64__)
00522 #  define ARCHITECTURE_ID "PPC64"
00523
00524 # elif defined(__ppc__)
00525 #  define ARCHITECTURE_ID "PPC"
00526
00527 # elif defined(__ARM__)
00528 #  define ARCHITECTURE_ID "ARM"
00529
00530 # elif defined(__x86_64__)
00531 #  define ARCHITECTURE_ID "x64"
00532
00533 # elif defined(__i386__)
00534 #  define ARCHITECTURE_ID "X86"
00535
00536 # else /* unknown architecture */
00537 #  define ARCHITECTURE_ID ""
00538 # endif
00539 #else
00540 #  define ARCHITECTURE_ID
00541 #endif
00542
00543 /* Convert integer to decimal digit literals.  */
00544 #define DEC(n)                   \
00545   ('0' + (((n) / 10000000)%10)), \
00546   ('0' + (((n) / 1000000)%10)),  \
00547   ('0' + (((n) / 100000)%10)),   \
00548   ('0' + (((n) / 10000)%10)),    \
00549   ('0' + (((n) / 1000)%10)),     \
00550   ('0' + (((n) / 100)%10)),      \
00551   ('0' + (((n) / 10)%10)),       \
00552   ('0' +  ((n) % 10))
00553
00554 /* Convert integer to hex digit literals.  */
00555 #define HEX(n)                   \
00556   ('0' + ((n)>>28 & 0xF)), \
00557   ('0' + ((n)>>24 & 0xF)), \
00558   ('0' + ((n)>>20 & 0xF)), \
00559   ('0' + ((n)>>16 & 0xF)), \
00560   ('0' + ((n)>>12 & 0xF)), \
00561   ('0' + ((n)>>8  & 0xF)), \
00562   ('0' + ((n)>>4  & 0xF)), \
00563   ('0' + ((n)     & 0xF))
00564
00565 /* Construct a string literal encoding the version number components. */
00566 #ifdef COMPILER_VERSION_MAJOR
00567 char const info_version[] = {
00568   'I', 'N', 'F', 'O', ':',
00569   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00570   COMPILER_VERSION_MAJOR,
00571 # ifdef COMPILER_VERSION_MINOR
00572   '.', COMPILER_VERSION_MINOR,
00573 #  ifdef COMPILER_VERSION_PATCH
00574    '.', COMPILER_VERSION_PATCH,
00575 #   ifdef COMPILER_VERSION_TWEAK
00576    '.', COMPILER_VERSION_TWEAK,
00577 #   endif
```

```
00578 #  endif
00579 # endif
00580   ']','\0'};
00581 #endif
00582
00583 /* Construct a string literal encoding the internal version number. */
00584 #ifdef COMPILER_VERSION_INTERNAL
00585 char const info_version_internal[] = {
00586   'I', 'N', 'F', 'O', ':',
00587   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00588   'i','n','t','e','r','n','a','l','[',
00589   COMPILER_VERSION_INTERNAL,']','\0'};
00590 #endif
00591
00592 /* Construct a string literal encoding the version number components. */
00593 #ifdef SIMULATE_VERSION_MAJOR
00594 char const info_simulate_version[] = {
00595   'I', 'N', 'F', 'O', ':',
00596   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00597   SIMULATE_VERSION_MAJOR,
00598 # ifdef SIMULATE_VERSION_MINOR
00599   '.', SIMULATE_VERSION_MINOR,
00600 #  ifdef SIMULATE_VERSION_PATCH
00601    '.', SIMULATE_VERSION_PATCH,
00602 #   ifdef SIMULATE_VERSION_TWEAK
00603     '.', SIMULATE_VERSION_TWEAK,
00604 #   endif
00605 #  endif
00606 # endif
00607   ']','\0'};
00608 #endif
00609
00610 /* Construct the string literal in pieces to prevent the source from
00611    getting matched.  Store it in a pointer rather than an array
00612    because some compilers will just produce instructions to fill the
00613    array rather than assigning a pointer to a static array.  */
00614 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00615 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00616
00617
00618
00619
00620 #if !defined(__STDC__)
00621 # if (defined(_MSC_VER) && !defined(__clang__)) \
00622   || (defined(__ibmxl__) || defined(__IBMC__))
00623 #  define C_DIALECT "90"
00624 # else
00625 #  define C_DIALECT
00626 # endif
00627 #elif __STDC_VERSION__ >= 201000L
00628 # define C_DIALECT "11"
00629 #elif __STDC_VERSION__ >= 199901L
00630 # define C_DIALECT "99"
00631 #else
00632 # define C_DIALECT "90"
00633 #endif
00634 const char* info_language_dialect_default =
00635   "INFO" ":" "dialect_default[" C_DIALECT "]";
00636
00637 /*--------------------------------------------------------------------------*/
00638
00639 #ifdef ID_VOID_MAIN
00640 void main() {}
00641 #else
00642 # if defined(__CLASSIC_C__)
00643 int main(argc, argv) int argc; char *argv[];
00644 # else
00645 int main(int argc, char* argv[])
00646 # endif
00647 {
00648   int require = 0;
00649   require += info_compiler[argc];
00650   require += info_platform[argc];
00651   require += info_arch[argc];
00652 #ifdef COMPILER_VERSION_MAJOR
00653   require += info_version[argc];
00654 #endif
00655 #ifdef COMPILER_VERSION_INTERNAL
00656   require += info_version_internal[argc];
00657 #endif
00658 #ifdef SIMULATE_ID
00659   require += info_simulate[argc];
00660 #endif
00661 #ifdef SIMULATE_VERSION_MAJOR
00662   require += info_simulate_version[argc];
00663 #endif
00664 #if defined(__CRAYXE) || defined(__CRAYXC)
```

```
00665   require += info_cray[argc];
00666 #endif
00667   require += info_language_dialect_default[argc];
00668   (void)argv;
00669   return require;
00670 }
00671 #endif
```

6.2 CMakeCXXCompilerId.cpp

```
00001 /* This source file must have a .cpp extension so that all C++ compilers
00002    recognize the extension without flags.  Borland does not know .cxx for
00003    example.  */
00004 #ifndef __cplusplus
00005 # error "A C compiler has been selected for C++."
00006 #endif
00007
00008
00009 /* Version number components: V=Version, R=Revision, P=Patch
00010    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00011
00012 #if defined(__COMO__)
00013 # define COMPILER_ID "Comeau"
00014   /* __COMO_VERSION__ = VRR */
00015 # define COMPILER_VERSION_MAJOR DEC(__COMO_VERSION__ / 100)
00016 # define COMPILER_VERSION_MINOR DEC(__COMO_VERSION__ % 100)
00017
00018 #elif defined(__INTEL_COMPILER) || defined(__ICC)
00019 # define COMPILER_ID "Intel"
00020 # if defined(_MSC_VER)
00021 #  define SIMULATE_ID "MSVC"
00022 # endif
00023 # if defined(__GNUC__)
00024 #  define SIMULATE_ID "GNU"
00025 # endif
00026   /* __INTEL_COMPILER = VRP */
00027 # define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00028 # define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00029 # if defined(__INTEL_COMPILER_UPDATE)
00030 #  define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00031 # else
00032 #  define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00033 # endif
00034 # if defined(__INTEL_COMPILER_BUILD_DATE)
00035   /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00036 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00037 # endif
00038 # if defined(_MSC_VER)
00039   /* _MSC_VER = VVRR */
00040 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00041 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00042 # endif
00043 # if defined(__GNUC__)
00044 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00045 # elif defined(__GNUG__)
00046 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00047 # endif
00048 # if defined(__GNUC_MINOR__)
00049 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00050 # endif
00051 # if defined(__GNUC_PATCHLEVEL__)
00052 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00053 # endif
00054
00055 #elif defined(__PATHCC__)
00056 # define COMPILER_ID "PathScale"
00057 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00058 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00059 # if defined(__PATHCC_PATCHLEVEL__)
00060 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00061 # endif
00062
00063 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00064 # define COMPILER_ID "Embarcadero"
00065 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00066 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00067 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00068
00069 #elif defined(__BORLANDC__)
00070 # define COMPILER_ID "Borland"
00071   /* __BORLANDC__ = 0xVRR */
00072 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00073 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
```

```
00074
00075 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00076 # define COMPILER_ID "Watcom"
00077   /* __WATCOMC__ = VVRR */
00078 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00079 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00080 # if (__WATCOMC__ % 10) > 0
00081 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00082 # endif
00083
00084 #elif defined(__WATCOMC__)
00085 # define COMPILER_ID "OpenWatcom"
00086   /* __WATCOMC__ = VVRP + 1100 */
00087 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00088 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00089 # if (__WATCOMC__ % 10) > 0
00090 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00091 # endif
00092
00093 #elif defined(__SUNPRO_CC)
00094 # define COMPILER_ID "SunPro"
00095 # if __SUNPRO_CC >= 0x5100
00096   /* __SUNPRO_CC = 0xVRRP */
00097 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»12)
00098 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xFF)
00099 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00100 # else
00101   /* __SUNPRO_CC = 0xVRP */
00102 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»8)
00103 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xF)
00104 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00105 # endif
00106
00107 #elif defined(__HP_aCC)
00108 # define COMPILER_ID "HP"
00109   /* __HP_aCC = VVRRPP */
00110 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00111 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00112 # define COMPILER_VERSION_PATCH DEC(__HP_aCC     % 100)
00113
00114 #elif defined(__DECCXX)
00115 # define COMPILER_ID "Compaq"
00116   /* __DECCXX_VER = VVRRTPPPP */
00117 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00118 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000  % 100)
00119 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER         % 10000)
00120
00121 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00122 # define COMPILER_ID "zOS"
00123   /* __IBMCPP__ = VRP */
00124 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00125 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00126 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00127
00128 #elif defined(__ibmxl__) && defined(__clang__)
00129 # define COMPILER_ID "XLClang"
00130 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00131 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00132 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00133 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00134
00135
00136 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00137 # define COMPILER_ID "XL"
00138   /* __IBMCPP__ = VRP */
00139 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00140 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00141 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00142
00143 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00144 # define COMPILER_ID "VisualAge"
00145   /* __IBMCPP__ = VRP */
00146 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00147 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00148 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00149
00150 #elif defined(__PGI)
00151 # define COMPILER_ID "PGI"
00152 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00153 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00154 # if defined(__PGIC_PATCHLEVEL__)
00155 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00156 # endif
00157
00158 #elif defined(_CRAYC)
00159 # define COMPILER_ID "Cray"
00160 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
```

```
00161 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00162
00163 #elif defined(__TI_COMPILER_VERSION__)
00164 # define COMPILER_ID "TI"
00165   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00166 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00167 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00168 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00169
00170 #elif defined(__FUJITSU) || defined(__FCC_VERSION) || defined(__fcc_version)
00171 # define COMPILER_ID "Fujitsu"
00172
00173 #elif defined(__ghs__)
00174 # define COMPILER_ID "GHS"
00175 /* __GHS_VERSION_NUMBER = VVVVRP */
00176 # ifdef __GHS_VERSION_NUMBER
00177 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00178 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00180 # endif
00181
00182 #elif defined(__SCO_VERSION__)
00183 # define COMPILER_ID "SCO"
00184
00185 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00186 # define COMPILER_ID "ARMCC"
00187 #if __ARMCC_VERSION >= 1000000
00188   /* __ARMCC_VERSION = VRRPPPP */
00189   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00190   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00191   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00192 #else
00193   /* __ARMCC_VERSION = VRPPPP */
00194   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00195   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00196   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00197 #endif
00198
00199
00200 #elif defined(__clang__) && defined(__apple_build_version__)
00201 # define COMPILER_ID "AppleClang"
00202 # if defined(_MSC_VER)
00203 #  define SIMULATE_ID "MSVC"
00204 # endif
00205 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00206 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00207 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00208 # if defined(_MSC_VER)
00209   /* _MSC_VER = VVRR */
00210 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00211 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00212 # endif
00213 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00214
00215 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00216 # define COMPILER_ID "ARMClang"
00217   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00218   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00219   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION     % 10000)
00220 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00221
00222 #elif defined(__clang__)
00223 # define COMPILER_ID "Clang"
00224 # if defined(_MSC_VER)
00225 #  define SIMULATE_ID "MSVC"
00226 # endif
00227 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00228 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00229 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00230 # if defined(_MSC_VER)
00231   /* _MSC_VER = VVRR */
00232 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00233 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00234 # endif
00235
00236 #elif defined(__GNUC__) || defined(__GNUG__)
00237 # define COMPILER_ID "GNU"
00238 # if defined(__GNUC__)
00239 #  define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00240 # else
00241 #  define COMPILER_VERSION_MAJOR DEC(__GNUG__)
00242 # endif
00243 # if defined(__GNUC_MINOR__)
00244 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00245 # endif
00246 # if defined(__GNUC_PATCHLEVEL__)
00247 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
```

```
00248 # endif
00249
00250 #elif defined(_MSC_VER)
00251 # define COMPILER_ID "MSVC"
00252   /* _MSC_VER = VVRR */
00253 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00254 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00255 # if defined(_MSC_FULL_VER)
00256 #  if _MSC_VER >= 1400
00257     /* _MSC_FULL_VER = VVRRPPPPP */
00258 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00259 #  else
00260     /* _MSC_FULL_VER = VVRRPPPP */
00261 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00262 #  endif
00263 # endif
00264 # if defined(_MSC_BUILD)
00265 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00266 # endif
00267
00268 #elif defined(__VISUALDSPVERSION__) || defined(__ADSPBLACKFIN__) || defined(__ADSPTS__) ||
       defined(__ADSP21000__)
00269 # define COMPILER_ID "ADSP"
00270 #if defined(__VISUALDSPVERSION__)
00271   /* __VISUALDSPVERSION__ = 0xVVRRPP00 */
00272 # define COMPILER_VERSION_MAJOR HEX(__VISUALDSPVERSION__»24)
00273 # define COMPILER_VERSION_MINOR HEX(__VISUALDSPVERSION__»16 & 0xFF)
00274 # define COMPILER_VERSION_PATCH HEX(__VISUALDSPVERSION__»8  & 0xFF)
00275 #endif
00276
00277 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00278 # define COMPILER_ID "IAR"
00279 # if defined(__VER__) && defined(__ICCARM__)
00280 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00281 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00282 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00283 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00284 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
       defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
       defined(__ICC8051__))
00285 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00286 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00287 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00288 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00289 # endif
00290
00291
00292 /* These compilers are either not known or too old to define an
00293   identification macro.  Try to identify the platform and guess that
00294   it is the native compiler.  */
00295 #elif defined(__hpux) || defined(__hpua)
00296 # define COMPILER_ID "HP"
00297
00298 #else /* unknown compiler */
00299 # define COMPILER_ID ""
00300 #endif
00301
00302 /* Construct the string literal in pieces to prevent the source from
00303    getting matched.  Store it in a pointer rather than an array
00304    because some compilers will just produce instructions to fill the
00305    array rather than assigning a pointer to a static array.  */
00306 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00307 #ifdef SIMULATE_ID
00308 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00309 #endif
00310
00311 #ifdef __QNXNTO__
00312 char const* qnxnto = "INFO" ":" "qnxnto[]";
00313 #endif
00314
00315 #if defined(__CRAYXE) || defined(__CRAYXC)
00316 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00317 #endif
00318
00319 #define STRINGIFY_HELPER(X) #X
00320 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00321
00322 /* Identify known platforms by name.  */
00323 #if defined(__linux) || defined(__linux__) || defined(linux)
00324 # define PLATFORM_ID "Linux"
00325
00326 #elif defined(__CYGWIN__)
00327 # define PLATFORM_ID "Cygwin"
00328
00329 #elif defined(__MINGW32__)
00330 # define PLATFORM_ID "MinGW"
00331
```

```
00332 #elif defined(__APPLE__)
00333 # define PLATFORM_ID "Darwin"
00334
00335 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00336 # define PLATFORM_ID "Windows"
00337
00338 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00339 # define PLATFORM_ID "FreeBSD"
00340
00341 #elif defined(__NetBSD__) || defined(__NetBSD)
00342 # define PLATFORM_ID "NetBSD"
00343
00344 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00345 # define PLATFORM_ID "OpenBSD"
00346
00347 #elif defined(__sun) || defined(sun)
00348 # define PLATFORM_ID "SunOS"
00349
00350 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00351 # define PLATFORM_ID "AIX"
00352
00353 #elif defined(__hpux) || defined(__hpux__)
00354 # define PLATFORM_ID "HP-UX"
00355
00356 #elif defined(__HAIKU__)
00357 # define PLATFORM_ID "Haiku"
00358
00359 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00360 # define PLATFORM_ID "BeOS"
00361
00362 #elif defined(__QNX__) || defined(__QNXNTO__)
00363 # define PLATFORM_ID "QNX"
00364
00365 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00366 # define PLATFORM_ID "Tru64"
00367
00368 #elif defined(__riscos) || defined(__riscos__)
00369 # define PLATFORM_ID "RISCos"
00370
00371 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00372 # define PLATFORM_ID "SINIX"
00373
00374 #elif defined(__UNIX_SV__)
00375 # define PLATFORM_ID "UNIX_SV"
00376
00377 #elif defined(__bsdos__)
00378 # define PLATFORM_ID "BSDOS"
00379
00380 #elif defined(_MPRAS) || defined(MPRAS)
00381 # define PLATFORM_ID "MP-RAS"
00382
00383 #elif defined(__osf) || defined(__osf__)
00384 # define PLATFORM_ID "OSF1"
00385
00386 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00387 # define PLATFORM_ID "SCO_SV"
00388
00389 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00390 # define PLATFORM_ID "ULTRIX"
00391
00392 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00393 # define PLATFORM_ID "Xenix"
00394
00395 #elif defined(__WATCOMC__)
00396 # if defined(__LINUX__)
00397 #  define PLATFORM_ID "Linux"
00398
00399 # elif defined(__DOS__)
00400 #  define PLATFORM_ID "DOS"
00401
00402 # elif defined(__OS2__)
00403 #  define PLATFORM_ID "OS2"
00404
00405 # elif defined(__WINDOWS__)
00406 #  define PLATFORM_ID "Windows3x"
00407
00408 # else /* unknown platform */
00409 #  define PLATFORM_ID
00410 # endif
00411
00412 #elif defined(__INTEGRITY)
00413 # if defined(INT_178B)
00414 #  define PLATFORM_ID "Integrity178"
00415
00416 # else /* regular Integrity */
00417 #  define PLATFORM_ID "Integrity"
00418 # endif
```

```
00419
00420 #else /* unknown platform */
00421 # define PLATFORM_ID
00422
00423 #endif
00424
00425 /* For windows compilers MSVC and Intel we can determine
00426    the architecture of the compiler being used.  This is because
00427    the compilers do not have flags that can change the architecture,
00428    but rather depend on which compiler is being used
00429 */
00430 #if defined(_WIN32) && defined(_MSC_VER)
00431 # if defined(_M_IA64)
00432 #  define ARCHITECTURE_ID "IA64"
00433
00434 # elif defined(_M_X64) || defined(_M_AMD64)
00435 #  define ARCHITECTURE_ID "x64"
00436
00437 # elif defined(_M_IX86)
00438 #  define ARCHITECTURE_ID "X86"
00439
00440 # elif defined(_M_ARM64)
00441 #  define ARCHITECTURE_ID "ARM64"
00442
00443 # elif defined(_M_ARM)
00444 #  if _M_ARM == 4
00445 #   define ARCHITECTURE_ID "ARMV4I"
00446 #  elif _M_ARM == 5
00447 #   define ARCHITECTURE_ID "ARMV5I"
00448 #  else
00449 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00450 #  endif
00451
00452 # elif defined(_M_MIPS)
00453 #  define ARCHITECTURE_ID "MIPS"
00454
00455 # elif defined(_M_SH)
00456 #  define ARCHITECTURE_ID "SHx"
00457
00458 # else /* unknown architecture */
00459 #  define ARCHITECTURE_ID ""
00460 # endif
00461
00462 #elif defined(__WATCOMC__)
00463 # if defined(_M_I86)
00464 #  define ARCHITECTURE_ID "I86"
00465
00466 # elif defined(_M_IX86)
00467 #  define ARCHITECTURE_ID "X86"
00468
00469 # else /* unknown architecture */
00470 #  define ARCHITECTURE_ID ""
00471 # endif
00472
00473 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00474 # if defined(__ICCARM__)
00475 #  define ARCHITECTURE_ID "ARM"
00476
00477 # elif defined(__ICCRX__)
00478 #  define ARCHITECTURE_ID "RX"
00479
00480 # elif defined(__ICCRH850__)
00481 #  define ARCHITECTURE_ID "RH850"
00482
00483 # elif defined(__ICCRL78__)
00484 #  define ARCHITECTURE_ID "RL78"
00485
00486 # elif defined(__ICCRISCV__)
00487 #  define ARCHITECTURE_ID "RISCV"
00488
00489 # elif defined(__ICCAVR__)
00490 #  define ARCHITECTURE_ID "AVR"
00491
00492 # elif defined(__ICC430__)
00493 #  define ARCHITECTURE_ID "MSP430"
00494
00495 # elif defined(__ICCV850__)
00496 #  define ARCHITECTURE_ID "V850"
00497
00498 # elif defined(__ICC8051__)
00499 #  define ARCHITECTURE_ID "8051"
00500
00501 # else /* unknown architecture */
00502 #  define ARCHITECTURE_ID ""
00503 # endif
00504
00505 #elif defined(__ghs__)
```

```
00506 # if defined(__PPC64__)
00507 #  define ARCHITECTURE_ID "PPC64"
00508
00509 # elif defined(__ppc__)
00510 #  define ARCHITECTURE_ID "PPC"
00511
00512 # elif defined(__ARM__)
00513 #  define ARCHITECTURE_ID "ARM"
00514
00515 # elif defined(__x86_64__)
00516 #  define ARCHITECTURE_ID "x64"
00517
00518 # elif defined(__i386__)
00519 #  define ARCHITECTURE_ID "X86"
00520
00521 # else /* unknown architecture */
00522 #  define ARCHITECTURE_ID ""
00523 # endif
00524 #else
00525 #  define ARCHITECTURE_ID
00526 #endif
00527
00528 /* Convert integer to decimal digit literals.  */
00529 #define DEC(n)                   \
00530   ('0' + (((n) / 10000000)%10)), \
00531   ('0' + (((n) / 1000000)%10)),  \
00532   ('0' + (((n) / 100000)%10)),   \
00533   ('0' + (((n) / 10000)%10)),    \
00534   ('0' + (((n) / 1000)%10)),     \
00535   ('0' + (((n) / 100)%10)),      \
00536   ('0' + (((n) / 10)%10)),       \
00537   ('0' +  ((n) % 10))
00538
00539 /* Convert integer to hex digit literals.  */
00540 #define HEX(n)             \
00541   ('0' + ((n)>>28 & 0xF)), \
00542   ('0' + ((n)>>24 & 0xF)), \
00543   ('0' + ((n)>>20 & 0xF)), \
00544   ('0' + ((n)>>16 & 0xF)), \
00545   ('0' + ((n)>>12 & 0xF)), \
00546   ('0' + ((n)>>8  & 0xF)), \
00547   ('0' + ((n)>>4  & 0xF)), \
00548   ('0' + ((n)     & 0xF))
00549
00550 /* Construct a string literal encoding the version number components. */
00551 #ifdef COMPILER_VERSION_MAJOR
00552 char const info_version[] = {
00553   'I', 'N', 'F', 'O', ':',
00554   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00555   COMPILER_VERSION_MAJOR,
00556 # ifdef COMPILER_VERSION_MINOR
00557   '.', COMPILER_VERSION_MINOR,
00558 #  ifdef COMPILER_VERSION_PATCH
00559    '.', COMPILER_VERSION_PATCH,
00560 #   ifdef COMPILER_VERSION_TWEAK
00561    '.', COMPILER_VERSION_TWEAK,
00562 #   endif
00563 #  endif
00564 # endif
00565   ']','\0'};
00566 #endif
00567
00568 /* Construct a string literal encoding the internal version number. */
00569 #ifdef COMPILER_VERSION_INTERNAL
00570 char const info_version_internal[] = {
00571   'I', 'N', 'F', 'O', ':',
00572   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00573   'i','n','t','e','r','n','a','l','[',
00574   COMPILER_VERSION_INTERNAL,']','\0'};
00575 #endif
00576
00577 /* Construct a string literal encoding the version number components. */
00578 #ifdef SIMULATE_VERSION_MAJOR
00579 char const info_simulate_version[] = {
00580   'I', 'N', 'F', 'O', ':',
00581   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00582   SIMULATE_VERSION_MAJOR,
00583 # ifdef SIMULATE_VERSION_MINOR
00584   '.', SIMULATE_VERSION_MINOR,
00585 #  ifdef SIMULATE_VERSION_PATCH
00586    '.', SIMULATE_VERSION_PATCH,
00587 #   ifdef SIMULATE_VERSION_TWEAK
00588    '.', SIMULATE_VERSION_TWEAK,
00589 #   endif
00590 #  endif
00591 # endif
00592   ']','\0'};
```

```
00593 #endif
00594
00595 /* Construct the string literal in pieces to prevent the source from
00596    getting matched.  Store it in a pointer rather than an array
00597    because some compilers will just produce instructions to fill the
00598    array rather than assigning a pointer to a static array.  */
00599 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00600 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00601
00602
00603
00604
00605 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG) && _MSVC_LANG < 201403L
00606 #  if defined(__INTEL_CXX11_MODE__)
00607 #    if defined(__cpp_aggregate_nsdmi)
00608 #      define CXX_STD 201402L
00609 #    else
00610 #      define CXX_STD 201103L
00611 #    endif
00612 #  else
00613 #    define CXX_STD 199711L
00614 #  endif
00615 #elif defined(_MSC_VER) && defined(_MSVC_LANG)
00616 #  define CXX_STD _MSVC_LANG
00617 #else
00618 #  define CXX_STD __cplusplus
00619 #endif
00620
00621 const char* info_language_dialect_default = "INFO" ":" "dialect_default["
00622 #if CXX_STD > 201703L
00623   "20"
00624 #elif CXX_STD >= 201703L
00625   "17"
00626 #elif CXX_STD >= 201402L
00627   "14"
00628 #elif CXX_STD >= 201103L
00629   "11"
00630 #else
00631   "98"
00632 #endif
00633 "]";
00634
00635 /*--------------------------------------------------------------------------*/
00636
00637 int main(int argc, char* argv[])
00638 {
00639   int require = 0;
00640   require += info_compiler[argc];
00641   require += info_platform[argc];
00642 #ifdef COMPILER_VERSION_MAJOR
00643   require += info_version[argc];
00644 #endif
00645 #ifdef COMPILER_VERSION_INTERNAL
00646   require += info_version_internal[argc];
00647 #endif
00648 #ifdef SIMULATE_ID
00649   require += info_simulate[argc];
00650 #endif
00651 #ifdef SIMULATE_VERSION_MAJOR
00652   require += info_simulate_version[argc];
00653 #endif
00654 #if defined(__CRAYXE) || defined(__CRAYXC)
00655   require += info_cray[argc];
00656 #endif
00657   require += info_language_dialect_default[argc];
00658   (void)argv;
00659   return require;
00660 }
```

6.3 np.hpp

```
00001 #ifndef NP_H_
00002 #define NP_H_
00003
00004 #include "boost/multi_array.hpp"
00005 #include "boost/array.hpp"
00006 #include "boost/cstdlib.hpp"
00007 #include <type_traits>
00008 #include <cassert>
00009 #include <iostream>
00010 #include <functional>
00011
00018 namespace np
```

```
00019 {
00020
00021     typedef double ndArrayValue;
00022
00024     template <std::size_t ND>
00025     inline boost::multi_array<ndArrayValue, ND>::index
00026     getIndex(const boost::multi_array<ndArrayValue, ND> &m, const ndArrayValue *requestedElement,
      const unsigned short int direction)
00027     {
00028         int offset = requestedElement - m.origin();
00029         return (offset / m.strides()[direction] % m.shape()[direction] + m.index_bases()[direction]);
00030     }
00031
00033     template <std::size_t ND>
00034     inline boost::array<typename boost::multi_array<ndArrayValue, ND>::index, ND>
00035     getIndexArray(const boost::multi_array<ndArrayValue, ND> &m, const ndArrayValue *requestedElement)
00036     {
00037         using indexType = boost::multi_array<ndArrayValue, ND>::index;
00038         boost::array<indexType, ND> _index;
00039         for (unsigned int dir = 0; dir < ND; dir++)
00040         {
00041             _index[dir] = getIndex(m, requestedElement, dir);
00042         }
00043
00044         return _index;
00045     }
00046
00049     template <typename Array, typename Element, typename Functor>
00050     inline void for_each(const boost::type<Element> &type_dispatch,
00051                          Array A, Functor &xform)
00052     {
00053         for_each(type_dispatch, A.begin(), A.end(), xform);
00054     }
00055
00057     template <typename Element, typename Functor>
00058     inline void for_each(const boost::type<Element> &, Element &Val, Functor &xform)
00059     {
00060         Val = xform(Val);
00061     }
00062
00064     template <typename Element, typename Iterator, typename Functor>
00065     inline void for_each(const boost::type<Element> &type_dispatch,
00066                          Iterator begin, Iterator end,
00067                          Functor &xform)
00068     {
00069         while (begin != end)
00070         {
00071             for_each(type_dispatch, *begin, xform);
00072             ++begin;
00073         }
00074     }
00075
00078     template <typename Array, typename Functor>
00079     inline void for_each(Array &A, Functor xform)
00080     {
00081         // Dispatch to the proper function
00082         for_each(boost::type<typename Array::element>(), A.begin(), A.end(), xform);
00083     }
00084
00088     template <long unsigned int ND>
00089     inline constexpr std::vector<boost::multi_array<double, ND» gradient(boost::multi_array<double,
      ND> inArray, std::initializer_list<double> args)
00090     {
00091         // static_assert(args.size() == ND, "Number of arguments must match the number of dimensions
      of the array");
00092         using arrayIndex = boost::multi_array<double, ND>::index;
00093
00094         using ndIndexArray = boost::array<arrayIndex, ND>;
00095
00096         // constexpr std::size_t n = sizeof...(Args);
00097         std::size_t n = args.size();
00098         // std::tuple<Args...> store(args...);
00099         std::vector<double> arg_vector = args;
00100         boost::multi_array<double, ND> my_array;
00101         std::vector<boost::multi_array<double, ND» output_arrays;
00102         for (std::size_t i = 0; i < n; i++)
00103         {
00104             boost::multi_array<double, ND> dfdh = inArray;
00105             output_arrays.push_back(dfdh);
00106         }
00107
00108         ndArrayValue *p = inArray.data();
00109         ndIndexArray index;
00110         for (std::size_t i = 0; i < inArray.num_elements(); i++)
00111         {
00112             index = getIndexArray(inArray, p);
00113             /*
```

```
00114             std::cout « "Index: ";
00115             for (std::size_t j = 0; j < n; j++)
00116             {
00117                 std::cout « index[j] « " ";
00118             }
00119             std::cout « "\n";
00120             */
00121             // Calculating the gradient now
00122             // j is the axis/dimension
00123             for (std::size_t j = 0; j < n; j++)
00124             {
00125                 ndIndexArray index_high = index;
00126                 double dh_high;
00127                 if ((long unsigned int)index_high[j] < inArray.shape()[j] - 1)
00128                 {
00129                     index_high[j] += 1;
00130                     dh_high = arg_vector[j];
00131                 }
00132                 else
00133                 {
00134                     dh_high = 0;
00135                 }
00136                 ndIndexArray index_low = index;
00137                 double dh_low;
00138                 if (index_low[j] > 0)
00139                 {
00140                     index_low[j] -= 1;
00141                     dh_low = arg_vector[j];
00142                 }
00143                 else
00144                 {
00145                     dh_low = 0;
00146                 }
00147 
00148                 double dh = dh_high + dh_low;
00149                 double gradient = (inArray(index_high) - inArray(index_low)) / dh;
00150                 // std::cout « gradient « "\n";
00151                 output_arrays[j](index) = gradient;
00152             }
00153             // std::cout « " value = " « inArray(index) « "  check = " « *p « std::endl;
00154             ++p;
00155         }
00156         return output_arrays;
00157     }
00158 
00160     inline boost::multi_array<double, 1> linspace(double start, double stop, long unsigned int num)
00161     {
00162         double step = (stop - start) / (num - 1);
00163         boost::multi_array<double, 1> output(boost::extents[num]);
00164         for (std::size_t i = 0; i < num; i++)
00165         {
00166             output[i] = start + i * step;
00167         }
00168         return output;
00169     }
00170 
00173     inline boost::multi_array<double, 1> zeros(long unsigned int num)
00174     {
00175         boost::multi_array<double, 1> output(boost::extents[num]);
00176         for (std::size_t i = 0; i < num; i++)
00177         {
00178             output[i] = 0;
00179         }
00180         return output;
00181     }
00182 
00183     enum indexing
00184     {
00185         xy,
00186         ij
00187     };
00188 
00194     template <long unsigned int ND>
00195     inline std::vector<boost::multi_array<double, ND» meshgrid(const boost::multi_array<double, 1>
        (&cinput)[ND], bool sparsing = false, indexing indexing_type = xy)
00196     {
00197         using arrayIndex = boost::multi_array<double, ND>::index;
00198         using ndIndexArray = boost::array<arrayIndex, ND>;
00199         std::vector<boost::multi_array<double, ND» output_arrays;
00200         boost::multi_array<double, 1> ci[ND];
00201         // Copy elements of cinput to ci, do the proper inversions
00202         for (std::size_t i = 0; i < ND; i++)
00203         {
00204             std::size_t source = i;
00205             if (indexing_type == xy && (ND == 3 || ND == 2))
00206             {
00207                 switch (i)
```

```
00208                 {
00209                 case 0:
00210                     source = 1;
00211                     break;
00212                 case 1:
00213                     source = 0;
00214                     break;
00215                 default:
00216                     break;
00217                 }
00218             }
00219             ci[i] = boost::multi_array<double, 1>();
00220             ci[i].resize(boost::extents[cinput[source].num_elements()]);
00221             ci[i] = cinput[source];
00222         }
00223         // Deducing the extents of the N-Dimensional output
00224         boost::detail::multi_array::extent_gen<ND> output_extents;
00225         std::vector<size_t> shape_list;
00226         for (std::size_t i = 0; i < ND; i++)
00227         {
00228             shape_list.push_back(ci[i].shape()[0]);
00229         }
00230         std::copy(shape_list.begin(), shape_list.end(), output_extents.ranges_.begin());
00231 
00232         // Creating the output arrays
00233         for (std::size_t i = 0; i < ND; i++)
00234         {
00235             boost::multi_array<double, ND> output_array(output_extents);
00236             ndArrayValue *p = output_array.data();
00237             ndIndexArray index;
00238             // Looping through the elements of the output array
00239             for (std::size_t j = 0; j < output_array.num_elements(); j++)
00240             {
00241                 index = getIndexArray(output_array, p);
00242                 boost::multi_array<double, 1>::index index_1d;
00243                 index_1d = index[i];
00244                 output_array(index) = ci[i][index_1d];
00245                 ++p;
00246             }
00247             output_arrays.push_back(output_array);
00248         }
00249         return output_arrays;
00250     }
00251 
00253     template <class T, long unsigned int ND>
00254     inline boost::multi_array<T, ND> element_wise_apply(const boost::multi_array<T, ND> &input_array,
    std::function<T(T)> func)
00255     {
00256 
00257         // Create output array copying extents
00258         using arrayIndex = boost::multi_array<double, ND>::index;
00259         using ndIndexArray = boost::array<arrayIndex, ND>;
00260         boost::detail::multi_array::extent_gen<ND> output_extents;
00261         std::vector<size_t> shape_list;
00262         for (std::size_t i = 0; i < ND; i++)
00263         {
00264             shape_list.push_back(input_array.shape()[i]);
00265         }
00266         std::copy(shape_list.begin(), shape_list.end(), output_extents.ranges_.begin());
00267         boost::multi_array<T, ND> output_array(output_extents);
00268 
00269         // Looping through the elements of the output array
00270         const T *p = input_array.data();
00271         ndIndexArray index;
00272         for (std::size_t i = 0; i < input_array.num_elements(); i++)
00273         {
00274             index = getIndexArray(input_array, p);
00275             output_array(index) = func(input_array(index));
00276             ++p;
00277         }
00278         return output_array;
00279     }
00280 
00281     // Complex operations
00282 
00283     template <class T, long unsigned int ND>
00284     inline boost::multi_array<T, ND> sqrt(const boost::multi_array<T, ND> &input_array)
00285     {
00286         std::function<T(T)> func = (T(*)(T))std::sqrt;
00287         return element_wise_apply(input_array, func);
00288     }
00289     template <class T>
00290     inline T sqrt(const T input)
00291     {
00292         return std::sqrt(input);
00293     }
00294 
```

```
00295     template <class T, long unsigned int ND>
00296     inline boost::multi_array<T, ND> exp(const boost::multi_array<T, ND> &input_array)
00297     {
00298         std::function<T(T)> func = (T(*)(T))std::exp;
00299         return element_wise_apply(input_array, func);
00300     }
00301     template <class T>
00302     inline T exp(const T input)
00303     {
00304         return std::exp(input);
00305     }
00306
00307     template <class T, long unsigned int ND>
00308     inline boost::multi_array<T, ND> log(const boost::multi_array<T, ND> &input_array)
00309     {
00310         std::function<T(T)> func = std::log<T>();
00311         return element_wise_apply(input_array, func);
00312     }
00313     template <class T>
00314     inline T log(const T input)
00315     {
00316         return std::log(input);
00317     }
00318     template <class T, long unsigned int ND>
00319     inline boost::multi_array<T, ND> pow(const boost::multi_array<T, ND> &input_array, const T
    exponent)
00320     {
00321         std::function<T(T)> pow_func = [exponent](T input)
00322         { return std::pow(input, exponent); };
00323         return element_wise_apply(input_array, pow_func);
00324     }
00325     template <class T>
00326     inline T pow(const T input, const T exponent)
00327     {
00328         return std::pow(input, exponent);
00329     }
00330
00334     template <class T, long unsigned int ND>
00335     boost::multi_array<T, ND> element_wise_duo_apply(boost::multi_array<T, ND> const &lhs,
    boost::multi_array<T, ND> const &rhs, std::function<T(T, T)> func)
00336     {
00337         // Create output array copying extents
00338         using arrayIndex = boost::multi_array<double, ND>::index;
00339         using ndIndexArray = boost::array<arrayIndex, ND>;
00340         boost::detail::multi_array::extent_gen<ND> output_extents;
00341         std::vector<size_t> shape_list;
00342         for (std::size_t i = 0; i < ND; i++)
00343         {
00344             shape_list.push_back(lhs.shape()[i]);
00345         }
00346         std::copy(shape_list.begin(), shape_list.end(), output_extents.ranges_.begin());
00347         boost::multi_array<T, ND> output_array(output_extents);
00348
00349         // Looping through the elements of the output array
00350         const T *p = lhs.data();
00351         ndIndexArray index;
00352         for (std::size_t i = 0; i < lhs.num_elements(); i++)
00353         {
00354             index = getIndexArray(lhs, p);
00355             output_array(index) = func(lhs(index), rhs(index));
00356             ++p;
00357         }
00358         return output_array;
00359     }
00360 }
00361
00362 // Basic operators
00363
00364 template <class T, long unsigned int ND>
00365 inline boost::multi_array<T, ND> operator*(boost::multi_array<T, ND> const &lhs, boost::multi_array<T,
    ND> const &rhs)
00366 {
00367     std::function<T(T, T)> func = std::multiplies<T>();
00368     return np::element_wise_duo_apply(lhs, rhs, func);
00369 }
00370 template <class T, long unsigned int ND>
00371 boost::multi_array<T, ND> operator+(boost::multi_array<T, ND> const &lhs, boost::multi_array<T, ND>
    const &rhs)
00372 {
00373     std::function<T(T, T)> func = std::plus<T>();
00374     return np::element_wise_duo_apply(lhs, rhs, func);
00375 }
00376 template <class T, long unsigned int ND>
00377 boost::multi_array<T, ND> operator-(boost::multi_array<T, ND> const &lhs, boost::multi_array<T, ND>
    const &rhs)
00378 {
00379     std::function<T(T, T)> func = std::minus<T>();
```

```
00380     return np::element_wise_duo_apply(lhs, rhs, func);
00381 }
00382 template <class T, long unsigned int ND>
00383 boost::multi_array<T, ND> operator/(boost::multi_array<T, ND> const &lhs, boost::multi_array<T, ND>
      const &rhs)
00384 {
00385     std::function<T(T, T)> func = std::divides<T>();
00386     return np::element_wise_duo_apply(lhs, rhs, func);
00387 }
00389 #endif
```

6.4 main.cpp

```
00001 #include <iostream>
00002 #include <string>
00003 #include "ExternalLibraries/cxxopts.hpp"
00004 #include "CustomLibraries/np.hpp"
00005
00006 // Command line arguments
00007 cxxopts::Options options("WaveSimC", "A wave propagation simulator written in C++ for seismic data
      processing.");
00008 int main(int argc, char *argv[])
00009 {
00010     // Parse command line arguments
00011     options.add_options()("d,debug", "Enable debugging")("i,input_file", "Input file path",
      cxxopts::value<std::string>())("o,output_file", "Output file path",
      cxxopts::value<std::string>())("v,verbose", "Verbose output",
      cxxopts::value<bool>()->default_value("false"));
00012     auto result = options.parse(argc, argv);
00013
00014     std::cout « "Hello World"
00015               « "\n";
00016 }
```

6.5 FiniteDifferenceWaveSolvers.cpp

```
00001 import <vector>;
00002
00003 export FiniteDifferenceWaveSolvers;
00004
00005 std::vector<double> np_gradient(std::vector<double> f, std::vector<double> dx, std::vector<double> dz)
00006 {
00007     std::vector<double> dfdx, dfdz;
00008     dfdx, dfdz = np.gradient(f, dx, dz) return dfdx, dfdz
00009 }
00010
00011 std::vector<double> dfdx(std::vector<double> fx, std::vector<double> dx) : dfdx, _ = np.gradient(fx,
      dx, dz) return dfdx
00012
00013                                                                                 def dfdz(fz, dz)
      : _,
00014                                                                           dfdz =
      np.gradient(fz, dx, dz) return dfdz
00015
00016                                                                                  def
      divergence(fx, fz, dx, dz) : dfdx,
00017                                                                           _ = np.gradient(fx,
      dx, dz)
00018                                                                                   _,
00019                                                                           dfdz =
      np.gradient(fz, dx, dz) return dfdx + dfdz
00020
00021                                                                                  def
      laplacian(f, dx, dz) : dfdx,
00022                                                                           dfdz = np.gradient(f,
      dx, dz)
00023                                                                                      d2fdx2,
00024                                                                           d2fdxdz =
      np.gradient(dfdx, dx, dz)
00025
      d2fdzdx,
00026                                                                           d2fdz2 =
      np.gradient(dfdz, dx, dz) return d2fdx2 + d2fdz2
00027
00028                                                                                    def d2dt2(f,
      dt) : dfdt,
00029                                                                           _, _ = np.gradient(f,
      dt, dt, dt)
00030                                                                                      d2fdt2,
```

```
00031                                                                                                   _, _ =
      np.gradient(dfdt, dt, dt, dt) return d2fdt2
00032
00033                                                                                                           def dfdt(f,
      dt) : dfdt,
00034                                                                                                   _, _ = np.gradient(f,
      dt, dt, dt) return dfdt
00035
00036                                                                                                           def
      timer(start, end) : hours,
00037                                                                                                   rem = divmod(end -
      start, 3600)
00038                                                                                                       minutes,
00039                                                                                                   seconds = divmod(rem,
      60)
00040
      print("{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds))
```

6.6 variadic.cpp

```
00001 #include "boost/multi_array.hpp"
00002 #include "boost/array.hpp"
00003 #include "CustomLibraries/np.hpp"
00004 #include <cassert>
00005 #include <iostream>
00006
00007 void test_gradient()
00008 {
00009     // Create a 4D array that is 3 x 4 x 2 x 1
00010     typedef boost::multi_array<double, 4>::index index;
00011     boost::multi_array<double, 4> A(boost::extents[3][4][2][2]);
00012
00013     // Assign values to the elements
00014     int values = 0;
00015     for (index i = 0; i != 3; ++i)
00016         for (index j = 0; j != 4; ++j)
00017             for (index k = 0; k != 2; ++k)
00018                 for (index l = 0; l != 2; ++l)
00019                     A[i][j][k][l] = values++;
00020
00021     // Verify values
00022     int verify = 0;
00023     for (index i = 0; i != 3; ++i)
00024         for (index j = 0; j != 4; ++j)
00025             for (index k = 0; k != 2; ++k)
00026                 for (index l = 0; l != 2; ++l)
00027                     assert(A[i][j][k][l] == verify++);
00028
00029     double dx, dy, dz, dt;
00030     dx = 1.0;
00031     dy = 1.0;
00032     dz = 1.0;
00033     dt = 1.0;
00034     std::vector<boost::multi_array<double, 4» my_arrays = np::gradient(A, {dx, dy, dz, dt});
00035
00036     boost::multi_array<double, 1> x = np::linspace(0, 1, 5);
00037     std::vector<boost::multi_array<double, 1» gradf = np::gradient(x, {1.0});
00038     for (int i = 0; i < 5; i++)
00039     {
00040         std::cout « gradf[0][i] « ",";
00041     }
00042     std::cout « "\n";
00043     // np::print(std::cout, my_arrays[0]);
00044 }
00045
00046 void test_meshgrid()
00047 {
00048     boost::multi_array<double, 1> x = np::linspace(0, 1, 5);
00049     boost::multi_array<double, 1> y = np::linspace(0, 1, 5);
00050     boost::multi_array<double, 1> z = np::linspace(0, 1, 5);
00051     boost::multi_array<double, 1> t = np::linspace(0, 1, 5);
00052     const boost::multi_array<double, 1> axis[4] = {x, y, z, t};
00053     std::vector<boost::multi_array<double, 4» my_arrays = np::meshgrid(axis, false, np::xy);
00054     // np::print(std::cout, my_arrays[0]);
00055     int nx = 3;
00056     int ny = 2;
00057     boost::multi_array<double, 1> x2 = np::linspace(0, 1, nx);
00058     boost::multi_array<double, 1> y2 = np::linspace(0, 1, ny);
00059     const boost::multi_array<double, 1> axis2[2] = {x2, y2};
00060     std::vector<boost::multi_array<double, 2» my_arrays2 = np::meshgrid(axis2, false, np::xy);
00061     std::cout « "xv\n";
00062     for (int i = 0; i < ny; i++)
00063     {
```

```
00064         for (int j = 0; j < nx; j++)
00065         {
00066             std::cout « my_arrays2[0][i][j] « " ";
00067         }
00068         std::cout « "\n";
00069     }
00070     std::cout « "yv\n";
00071     for (int i = 0; i < ny; i++)
00072     {
00073         for (int j = 0; j < nx; j++)
00074         {
00075             std::cout « my_arrays2[1][i][j] « " ";
00076         }
00077         std::cout « "\n";
00078     }
00079 }
00080
00081 void test_complex_operations()
00082 {
00083     int nx = 3;
00084     int ny = 2;
00085     boost::multi_array<double, 1> x = np::linspace(0, 1, nx);
00086     boost::multi_array<double, 1> y = np::linspace(0, 1, ny);
00087     const boost::multi_array<double, 1> axis[2] = {x, y};
00088     std::vector<boost::multi_array<double, 2» my_arrays = np::meshgrid(axis, false, np::xy);
00089     boost::multi_array<double, 2> A = np::sqrt(my_arrays[0]);
00090     std::cout « "sqrt\n";
00091     for (int i = 0; i < ny; i++)
00092     {
00093         for (int j = 0; j < nx; j++)
00094         {
00095             std::cout « A[i][j] « " ";
00096         }
00097         std::cout « "\n";
00098     }
00099     std::cout « "\n";
00100     float a = 100.0;
00101     float sqa = np::sqrt(a);
00102     std::cout « "sqrt of " « a « " is " « sqa « "\n";
00103     std::cout « "exp\n";
00104     boost::multi_array<double, 2> B = np::exp(my_arrays[0]);
00105     for (int i = 0; i < ny; i++)
00106     {
00107         for (int j = 0; j < nx; j++)
00108         {
00109             std::cout « B[i][j] « " ";
00110         }
00111         std::cout « "\n";
00112     }
00113
00114     std::cout « "Power\n";
00115     boost::multi_array<double, 1> x2 = np::linspace(1, 3, nx);
00116     boost::multi_array<double, 1> y2 = np::linspace(1, 3, ny);
00117     const boost::multi_array<double, 1> axis2[2] = {x2, y2};
00118     std::vector<boost::multi_array<double, 2» my_arrays2 = np::meshgrid(axis2, false, np::xy);
00119     boost::multi_array<double, 2> C = np::pow(my_arrays2[1], 2.0);
00120     for (int i = 0; i < ny; i++)
00121     {
00122         for (int j = 0; j < nx; j++)
00123         {
00124             std::cout « C[i][j] « " ";
00125         }
00126         std::cout « "\n";
00127     }
00128 }
00129
00130 void test_equal()
00131 {
00132     boost::multi_array<double, 1> x = np::linspace(0, 1, 5);
00133     boost::multi_array<double, 1> y = np::linspace(0, 1, 5);
00134     boost::multi_array<double, 1> z = np::linspace(0, 1, 5);
00135     boost::multi_array<double, 1> t = np::linspace(0, 1, 5);
00136     const boost::multi_array<double, 1> axis[4] = {x, y, z, t};
00137     std::vector<boost::multi_array<double, 4» my_arrays = np::meshgrid(axis, false, np::xy);
00138     boost::multi_array<double, 1> x2 = np::linspace(0, 1, 5);
00139     boost::multi_array<double, 1> y2 = np::linspace(0, 1, 5);
00140     boost::multi_array<double, 1> z2 = np::linspace(0, 1, 5);
00141     boost::multi_array<double, 1> t2 = np::linspace(0, 1, 5);
00142     const boost::multi_array<double, 1> axis2[4] = {x2, y2, z2, t2};
00143     std::vector<boost::multi_array<double, 4» my_arrays2 = np::meshgrid(axis2, false, np::xy);
00144     std::cout « "equality test:\n";
00145     std::cout « (bool)(my_arrays == my_arrays2) « "\n";
00146 }
00147 void test_basic_operations()
00148 {
00149     int nx = 3;
00150     int ny = 2;
```

```
00151     boost::multi_array<double, 1> x = np::linspace(0, 1, nx);
00152     boost::multi_array<double, 1> y = np::linspace(0, 1, ny);
00153     const boost::multi_array<double, 1> axis[2] = {x, y};
00154     std::vector<boost::multi_array<double, 2» my_arrays = np::meshgrid(axis, false, np::xy);
00155 
00156     std::cout « "basic operations:\n";
00157 
00158     std::cout « "addition:\n";
00159     boost::multi_array<double, 2> A = my_arrays[0] + my_arrays[1];
00160 
00161     for (int i = 0; i < ny; i++)
00162     {
00163         for (int j = 0; j < nx; j++)
00164         {
00165             std::cout « A[i][j] « " ";
00166         }
00167         std::cout « "\n";
00168     }
00169 
00170     std::cout « "multiplication:\n";
00171     boost::multi_array<double, 2> B = my_arrays[0] * my_arrays[1];
00172 
00173     for (int i = 0; i < ny; i++)
00174     {
00175         for (int j = 0; j < nx; j++)
00176         {
00177             std::cout « B[i][j] « " ";
00178         }
00179         std::cout « "\n";
00180     }
00181 }
00182 
00183 int main()
00184 {
00185     test_gradient();
00186     test_meshgrid();
00187     test_complex_operations();
00188     test_equal();
00189     test_basic_operations();
00190 }
```