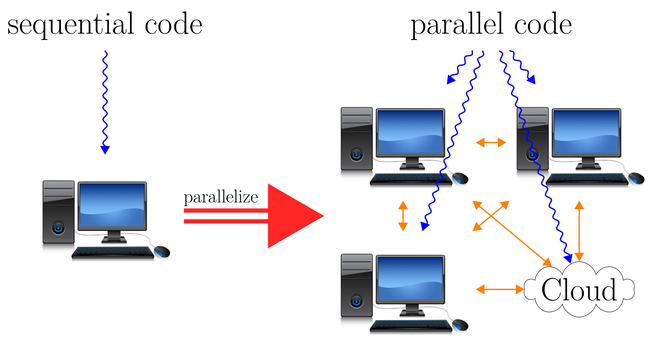
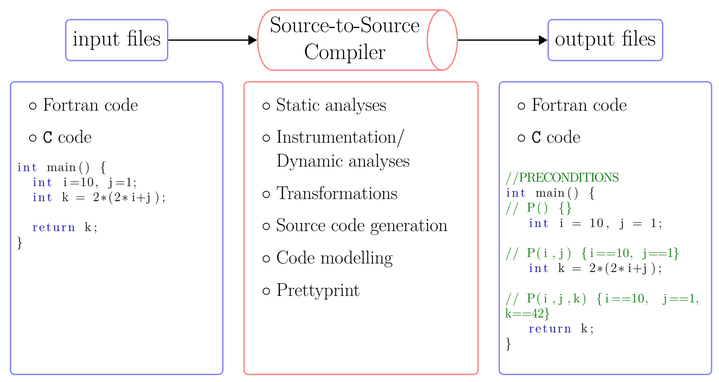
Compiler Passes for Automatic Task Distribution

Nelson Lossing
CRI, MINES ParisTech, PSL Research University

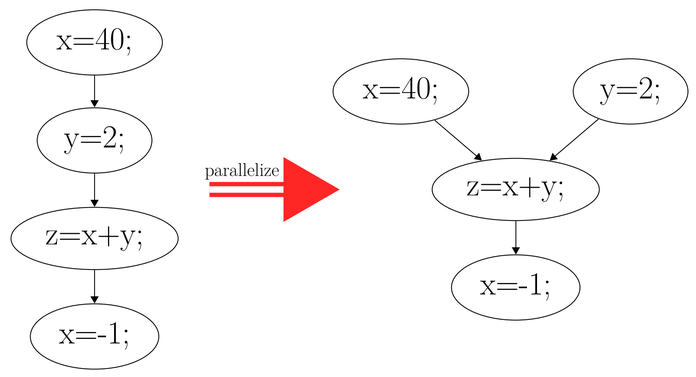
Motivations



PIPS



Task Scheduling



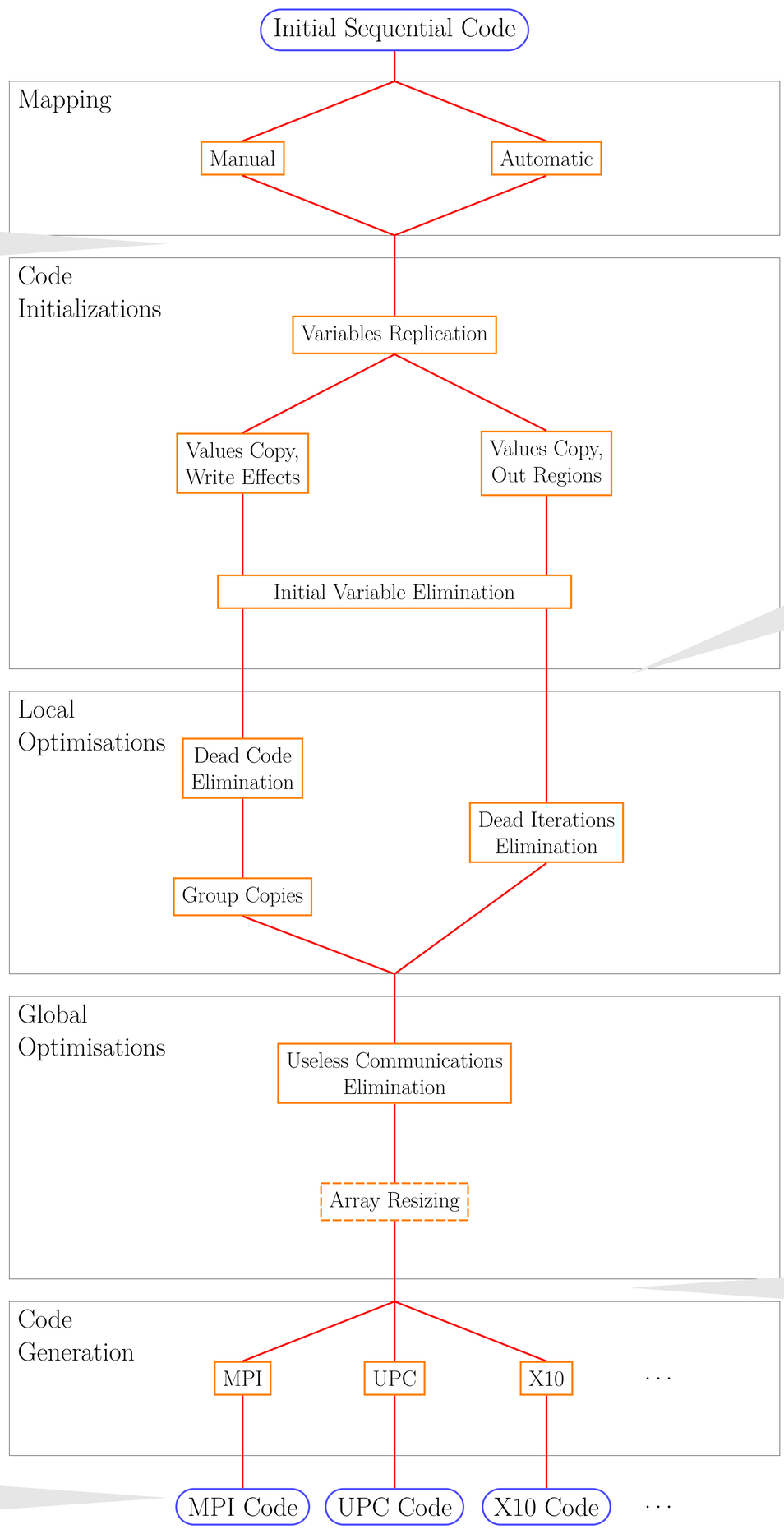
Initial Code

```
int main(int argc, char **argv) {
  /* problem size. */
  int ni = 1000, nj = 1100, nk = 1200;
  /* Variable declaration */
  double alpha, beta;
  double C[1000][1100], A[1000][1200], B[1200][1100];
  int i, j, k;

  /***** Initialization *****/
#pragma distributed on_cluster=0
  {
  init(ni, nj, nk, &alpha, &beta, C, A, B);
  }

  /***** Job to distribute *****/
#pragma distributed for
  {
  for(i = 0; i <= ni-1; i += 1)
    for(j = 0; j <= nj-1; j += 1)
      C[i][j] *= beta;
  }
#pragma distributed for
  {
  for(i = 0; i <= ni-1; i += 1)
    for(j = 0; j <= nj-1; j += 1)
      for(k = 0; k <= nk-1; k += 1)
        C[i][j] += alpha*A[i][k]*B[k][j];
  }

  /***** Print result *****/
#pragma distributed on_cluster=0
    print_array(ni, nj, c);
  return 0;
}
```

Final MPI Code Generated

```
int main(int argc, char **argv) {
  // Generated by Pass MPI_CONVERSION
  MPI_Status status0; MPI_Request request0;
  MPI_Init(&argc, &argv);
  // Generated by Pass VARIABLE_REPLICATION
  double _alpha_0, _alpha_1, _alpha_2, _alpha_3, _beta_0, ...;
  double _C_0[1000][1100], _C_1[1000][1100], ...;
  int i, j, k;
  /***** Initialization *****/
  if (rank0==0) {
    init(1000, 1100, 1200, &_alpha_0, &_beta_0, _C_0, _A_0, _B_0);
    // Generated by Pass COPY_VARIABLE with OUT-REGION
    MPI_Send(&_alpha_0, 1, MPI_DOUBLE, 1, 0, ...);
    MPI_Send(&_alpha_0, 1, MPI_DOUBLE, 2, 0, ...);
    MPI_Send(&_alpha_0, 1, MPI_DOUBLE, 3, 0, ...);
    //similar for _beta_0, _beta_1, _beta_2, _beta_3
    int _phi1_;
    for(_phi1_ = 250; _phi1_ <= 499; _phi1_ += 1)
        MPI_Send(&_C_0[_phi1_][0], 1100, MPI_DOUBLE, 1, 2, ...);
    for(_phi1_ = 500; _phi1_ <= 749; _phi1_ += 1)
        MPI_Send(&_C_0[_phi1_][0], 1100, MPI_DOUBLE, 2, 2, ...);
    for(_phi1_ = 750; _phi1_ <= 999; _phi1_ += 1)
        MPI_Send(&_C_0[_phi1_][0], 1100, MPI_DOUBLE, 3, 2, ...);
    //similar for _A_0, _A_1, _A_2, _A_3, _B_0, _B_1, _B_2, _B_3
  }
  if (rank0==1) {
    // Generated by Pass MPI_CONVERSION
    MPI_Recv(&_alpha_1, 1, MPI_DOUBLE, 0, 0, ...);
    MPI_Recv(&_beta_1, 1, MPI_DOUBLE, 0, 1, ...);
    int _phi1_;
    for(_phi1_ = 250; _phi1_ <= 499; _phi1_ += 1)
        MPI_Recv(&_C_1[_phi1_][0], 1100, MPI_DOUBLE, 0, 2, ...);
    //similar for _A_1, _B_1
  } ...
  /***** Job to distribute *****/
  if (rank0==0)
    for(i = 0; i <= 249; i += 1)
      for(j = 0; j <= 1099; j += 1)
        _C_0[i][j] *= _beta_0;
  ...
  if (rank0==0)
    for(i = 0; i <= 249; i += 1)
      for(j = 0; j <= 1099; j += 1)
        for(k = 0; k <= 1199; k += 1)
          _C_0[i][j] += _alpha_0*_A_0[i][k]*_B_0[k][j];
  if (rank0==1) {
    for(i = 250; i <= 499; i += 1)
      for(j = 0; j <= 1099; j += 1)
        for(k = 0; k <= 1199; k += 1)
          _C_1[i][j] += _alpha_1*_A_1[i][k]*_B_1[k][j];
    // Generated by Pass COPY_VARIABLE with OUT-REGION
    int _phi1_;
    for(_phi1_ = 250; _phi1_ <= 499; _phi1_ += 1)
        MPI_Send(&_C_1[_phi1_][0], 1100, MPI_DOUBLE, 0, 0, ...);
  }
  if (rank0==0) {
    // Generated by Pass MPI_CONVERSION
    int _phi1_;
    for(_phi1_ = 250; _phi1_ <= 499; _phi1_ += 1)
        MPI_Recv(&_C_0[_phi1_][0], 1100, MPI_DOUBLE, 1, 0, ...);
  }
  ...
  /***** Print result *****/
  if (rank0==0)
    print_array(1000, 1100, _C_0);
  MPI_Finalize();
  return 0;
}
```

Compilation Process



Communications as Copies

```
int main(int argc, char **argv) {
  // Generated by Pass VARIABLE_REPLICATION
  double _alpha_0, _alpha_1, _alpha_2, _alpha_3, _beta_0, ...;
  double _C_0[1000][1100], _C_1[1000][1100], ...;
  int i, j, k;
  /***** Initialization *****/
#pragma distributed on_cluster=0
  {
    init(1000, 1100, 1200, &_alpha_0, &_beta_0, _C_0, _A_0, _B_0);
    // Generated by Pass COPY_VARIABLE with OUT-REGION
    _alpha_1 = _alpha_0;
    _alpha_2 = _alpha_0;
    _alpha_3 = _alpha_0;
    //similar for _beta_0, _beta_1, _beta_2, _beta_3
    int _phi1_, _phi2_;
    for(_phi1_ = 0; _phi1_ <= 999; _phi1_ += 1)
      for(_phi2_ = 0; _phi2_ <= 1099; _phi2_ += 1) {
        _C_1[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
        _C_2[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
        _C_3[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
    }
    //similar for _A_0, _A_1, _A_2, _A_3, _B_0, _B_1, _B_2, _B_3
  }
  /***** Job to distribute *****/
#pragma distributed on_cluster=0
  {
    for(i = 0; i <= 249; i += 1)
      for(j = 0; j <= 1099; j += 1)
        _C_0[i][j] *= _beta_0;
    // Generated by Pass COPY_VARIABLE with OUT-REGION
    int _phi1_, _phi2_;
    for(_phi1_ = 0; _phi1_ <= 249; _phi1_ += 1)
      for(_phi2_ = 0; _phi2_ <= 1099; _phi2_ += 1) {
        _C_1[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
        _C_2[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
        _C_3[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
      }
  }
  ...
  /***** Print result *****/
#pragma distributed on_cluster=0
    print_array(1000, 1100, _C_0);
  return 0;
}
```

Removing Useless Copies-Communications

```
int main(int argc, char **argv) {
  // Generated by Pass VARIABLE_REPLICATION
  double _alpha_0, _alpha_1, _alpha_2, _alpha_3, _beta_0, ...;
  double _C_0[1000][1100], _C_1[1000][1100], ...;
  int i, j, k;
  /***** Initialization *****/
#pragma distributed on_cluster=0
  {
    init(1000, 1100, 1200, &_alpha_0, &_beta_0, _C_0, _A_0, _B_0);
    // Generated by Pass COPY_VARIABLE with OUT-REGION
    _alpha_1 = _alpha_0;
    _alpha_2 = _alpha_0;
    _alpha_3 = _alpha_0;
    //similar for _beta_0, _beta_1, _beta_2, _beta_3
    int _phi1_, _phi2_;
    for(_phi1_ = 250; _phi1_ <= 499; _phi1_ += 1)
      for(_phi2_ = 0; _phi2_ <= 1099; _phi2_ += 1)
        _C_1[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
    for(_phi1_ = 500; _phi1_ <= 749; _phi1_ += 1)
      for(_phi2_ = 0; _phi2_ <= 1099; _phi2_ += 1)
        _C_2[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
    for(_phi1_ = 750; _phi1_ <= 999; _phi1_ += 1)
      for(_phi2_ = 0; _phi2_ <= 1099; _phi2_ += 1)
        _C_3[_phi1_][_phi2_] = _C_0[_phi1_][_phi2_];
    //similar for _A_0, _A_1, _A_2, _A_3, _B_0, _B_1, _B_2, _B_3
  }
  /***** Job to distribute *****/
#pragma distributed on_cluster=0
  {
    for(i = 0; i <= 249; i += 1)
      for(j = 0; j <= 1099; j += 1)
        _C_0[i][j] *= _beta_0;
  }
  ...
#pragma distributed on_cluster=0
  {
    for(i = 0; i <= 249; i += 1)
      for(j = 0; j <= 1099; j += 1)
        for(k = 0; k <= 1199; k += 1)
          _C_0[i][j] += _alpha_0*_A_0[i][k]*_B_0[k][j];
  }
#pragma distributed on_cluster=1
  {
    for(i = 250; i <= 499; i += 1)
      for(j = 0; j <= 1099; j += 1)
        for(k = 0; k <= 1199; k += 1)
          _C_1[i][j] += _alpha_1*_A_1[i][k]*_B_1[k][j];
    // Generated by Pass COPY_VARIABLE with OUT-REGION
    int _phi1_, _phi2_;
    for(_phi1_ = 250; _phi1_ <= 499; _phi1_ += 1)
      for(_phi2_ = 0; _phi2_ <= 1099; _phi2_ += 1)
        _C_0[_phi1_][_phi2_] = _C_1[_phi1_][_phi2_];
  }
  ...
  /***** Print result *****/
#pragma distributed on_cluster=0
    print_array(1000, 1100, _C_0);
  return 0;
}
```

References

Dathathri, Roshan et al. "Generating efficient data movement code for heterogeneous architectures with distributed-memory". In: *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*. IEEE, 2013, pp. 375–386.

Khaldi, Dounia. "Automatic Resource-Constrained Static Task Parallelization : A Generic Approach". Theses. Ecole Nationale Supérieure des Mines de Paris, Nov. 2013.

Contributions

- Generation of Distributed Code from Sequential Code.
- Simple Step by Step Transformations.
- Elimination of Useless Communications.
- Optimisation of Communications.
- Optimisation of Local Memory.
- Semantically Equivalent Program.

Results