Appendix A: Source Code

vector.c

```
/*adds vectors a[] and b[] to yield s1[]
  and vectors a[] and c[] to yield s2 */
#define SIZE_N 10000000

int a[SIZE_N], b[SIZE_N], c[SIZE_N];
int s1[SIZE_N], s2[SIZE_N];

for (int i = 0; i < SIZE_N; i++)
    s1[i] = a[i] + b[i];

for (int i = 0; i < SIZE_N; i++)
    s2[i] = a[i] + c[i];
```

vector_fused.c

```
/*adds vectors a[] and b[] to yield s1[]
  and vectors a[] and c[] to yield s2
  with loop fusion */
#define SIZE_N 10000000

int a[SIZE_N], b[SIZE_N], c[SIZE_N];
int s1[SIZE_N], s2[SIZE_N];

for (int i = 0; i < SIZE_N; i++){
    s1[i] = a[i] + b[i];
    s2[i] = a[i] + c[i];
}
```

vector_prefetch.c

```
/*adds vectors a[] and b[] to yield s1[]
  and vectors a[] and c[] to yield s2
  with prefetch */
#define SIZE_N 10000000

int a[SIZE_N], b[SIZE_N], c[SIZE_N];
int s1[SIZE_N], s2[SIZE_N];

for (int i = 0; i < SIZE_N; i++){
    prefetch(a[i+50]);
    prefetch(b[i+50]);
    s1[i] = a[i] + b[i];
}

for (int i = 0; i < SIZE_N; i++){
    prefetch(a[i+50]);
    prefetch(c[i+50]);
    s2[i] = a[i] + c[i];
}
```

vector_fused_prefetch.c

```
/*adds vectors a[] and b[] to yield s1[]
  and vectors a[] and c[] to yield s2
  fused loops with prefetch */
#define SIZE_N 10000000

int a[SIZE_N], b[SIZE_N], c[SIZE_N];
int s1[SIZE_N], s2[SIZE_N];

for (int i = 0; i < SIZE_N; i++){
    prefetch(a[i+50]);
    prefetch(b[i+50]);
    prefetch(c[i+50]);
    s1[i] = a[i] + b[i];
    s2[i] = a[i] + c[i];
}
```

vector2.c

```
/*adds partially overlapping segments of
  vectors a[] and b[] to yield sums s1[] and s2[] */
#define SIZE_N 10000000

/* 0 <= START1 < START2 < END1 < END2 <= SIZE_N
#define START1 0
#define START2 1000000
#define END1 8000000
#define END2 10000000

int a[SIZE_N], b[SIZE_N];
int s1[END1 - START1 + 1], s2[END2 - START2 + 1];

for (int i = START1; i < END1; i++)
    s1[i - START1] = a[i] + b[i];

for (int i = START2; i < END2; i++)
    s2[i - START2] = a[i] + b[i];
```

vector2_fused.c

```
/*adds partially overlapping segments of
  vectors a[] and b[] to yield sums s1[] and s2[]
  with loop fusion */
#define SIZE_N 10000000

/* 0 <= START1 < START2 < END1 < END2 <= SIZE_N
#define START1 0
#define START2 1000000
#define END1 8000000
#define END2 10000000

int a[SIZE_N], b[SIZE_N];
int s1[END1 - START1 + 1], s2[END2 - START2 + 1];

for (int i = START1; i < START2; i++)
    s1[i - START1] = a[i] + b[i];

for (int i = START2; i < END1; i++){
    s1[i - START1] = a[i] + b[i];
    s2[i - START2] = a[i] + b[i];
}

for (int i = END1; i < END2; i++)
    s2[i - START2] = a[i] + b[i];
```

vector2_prefetch.c

```
/*adds partially overlapping segments of
  vectors a[] and b[] to yield sums s1[] and s2[] */
#define SIZE_N 10000000

/* 0 <= START1 < START2 < END1 < END2 <= SIZE_N
#define START1 0
#define START2 1000000
#define END1 8000000
#define END2 10000000

int a[SIZE_N], b[SIZE_N];
int s1[END1 - START1 + 1], s2[END2 - START2 + 1];

for (int i = START1; i < END1; i++){
    prefetch(a[i+50]);
    prefetch(b[i+50]);
    s1[i - START1] = a[i] + b[i];
}

for (int i = START2; i < END2; i++){
    prefetch(a[i+50]);
    prefetch(b[i+50]);
    s2[i - START2] = a[i] + b[i];
}
```

vector2_fused_prefetch.c

```
/*adds partially overlapping segments of
  vectors a[] and b[] to yield sums s1[] and s2[]
  with loop fusion and prefetch */
#define SIZE_N 10000000

/* 0 <= START1 < START2 < END1 < END2 <= SIZE_N
#define START1 0
#define START2 1000000
#define END1 8000000
#define END2 10000000

int a[SIZE_N], b[SIZE_N];
int s1[END1 - START1 + 1], s2[END2 - START2 + 1];

for (int i = START1; i < START2; i++){
    prefetch(a[i + 50]);
    prefetch(b[i + 50]);
    s1[i - START1] = a[i] + b[i];
}

for (int i = START2; i < END1; i++){
    prefetch(a[i + 50]);
    prefetch(b[i + 50]);
    s1[i - START1] = a[i] + b[i];
    s2[i - START2] = a[i] + b[i];
}

for (int i = END1; i < END2; i++){
    prefetch(a[i + 50]);
    prefetch(b[i + 50]);
    s2[i - START2] = a[i] + b[i];
}
```

vector3.c

```
/*adds every 10000th element of
  vectors a[] and b[] to yield s1[] */
#define SIZE_N 100000000

int a[SIZE_N], b[SIZE_N];
int s1[SIZE_N / 10000];

for (int i = 0; i < SIZE_N; i += 10000)
    s1[i] = a[i] + b[i];
```

vector3_merged.c

```
/*adds every 10000th element of
  vectors a[] and b[] to yield s1[]
  with merged arrays */
#define SIZE_N 100000000

struct vector_ab_type{
    int a;
    int b;
}
vector_ab[SIZE_N];
int s1[SIZE_N / 10000];

for (int i = 0; i < SIZE_N; i += 10000) s1[i] = vector_ab[i].a + vector_ab[i].a;
```

vector3_prefetch.c

```
/*adds every 10000th element of
  vectors a[] and b[] to yield s1[] */
#define SIZE_N 100000000

int a[SIZE_N], b[SIZE_N];
int s1[SIZE_N / 10000];

for (int i = 0; i < SIZE_N; i += 10000){
    prefetch(a[i+50]);
    prefetch(b[i+50]);
    s1[i] = a[i] + b[i];
}
```

vector3_merged_prefetch.c

```
/*adds every 10000th element of
  vectors a[] and b[] to yield s1[]
  with merged arrays */
#define SIZE_N 100000000

struct vector_ab_type{
    int a;
    int b;
}
vector_ab[SIZE_N];
int s1[SIZE_N / 10000];

for (int i = 0; i < SIZE_N; i += 10000){
    prefetch(vector_ab[i+50]);
    s1[i] = vector_ab[i].a + vector_ab[i].a;
}
```

linkedlist.c

```
/*copies elements out of a
  linked list into an array */

#define SIZE 10000000

struct linked{
    int data;
    *linked next;
}
*linked start;

/*code to fill list here
  assume a 10,000,000
  null-terminated list */

*linked temp = start;
int a[SIZE], index=0;
while (temp->next)
{
    a[index++] = temp->data;
    temp = temp->next;
}
```

linkedlist_prefetch.c

```
/*copies elements out of a
  linked list into an array
  with prefetching */

#define SIZE 10000000

struct linked{
    int data;
    *linked next;
}
*linked start;

/*code to fill list here
  assume a 10,000,000
  null-terminated list */

*linked temp = start, temp2 = start;
int a[SIZE], index=0;
for (int i = 0; i < 100; i++)
    temp2 = temp2->next;

while (temp->next){
    a[index++] = temp->data;
    temp = temp->next;

    /*be careful not to dereference
      NULL pointers */
    if (temp2->next){
        temp2 = temp2->next;
        prefetch(temp2->next);
    }
}
```

binarytree.c

```
/*searches a binary tree */

struct node{
    int data;
    *node left, right;
}
*node top;

/*cost function that returns a pointer
  to the next node and flags if it's
  done searching*/
*node next_node(*node current, int value, int & found);

/*code to fill tree here */

*node temp = top;
int search_value, found=0;

do{
    temp = next_node(temp, search_value, found);
}until (found);
```

binarytree_prefetch.c

```
/*searches a binary tree
  speculatively prefetches */

struct node{
    int data;
    *node left, right;
}
*node top;

/*cost function that returns a pointer
  to the next node and flags if it's
  done searching*/
*node next_node(*node current, int value, int & found);

/*code to fill tree here */

*node temp = top;
int search_value, found=0;

do{
    if (temp->left) prefetch(temp->left);
    if (temp->right) prefetch(temp->right);
    temp = next_node(temp, search_value, found);
}until (found);
```